Exhibit A

microunity

Euterpe MicroArchitecture

DRAFT

Date of last revision of content: REDACTED

Euterpe MicroArchitecture

Publication number 910-00006

Revision number 1.4

Copyright

Trademarks

Disclaimer

MicroUnity Systems Engineering, Inc.
255 Caspian Drive
Sunnyvale, California 94089-1015

(408) 734-8100 Fax: (408) 734-8136

Table of Contents

Chapter 1

Introduction

Compliance

Euterpe is a processor for the MicroUnity MediaComputer based on an implementation of the *Terpsichore Architecture*. It conforms with the specifications in *MicroUnity Terpsichore System Architecture* dated April 14, 1994.

Scope of this Book

This document is intended to describe:

1. The details of the Euterpe implementation of the architecture
2. The ways Euterpe is different from the Terpsichore specification
3. The areas of Euterpe design that are not mentioned by the architecture

This document should be used in conjuction with the *Terpsichore System Architecture* document to understand how Euterpe works.

The Basics

Euterpe is a multiprocessor on a chip—5 hardware-based cylinders of control share a single datapath, cache, DRAM interface, and I/O interfaces. Each thread of control is assigned to one time slot in a cycle of five slots. This is known as a *barrel processor*. In this way, contention between cylinders has been eliminated whenever possible. Some exceptions to this include the serialization of hardware exceptions and access to off-chip memory.

The barrel processing introduces two clock rates: that of the underlying datapath, and that of each cylinder. We sometimes call these *minor* and *major*, respectively; the minor clock time is called a *tick* and the major clock time a *cycle*. Nominally, a tick is 772ps and a cycle is 5 times that, or 3.86ns.

General Registers

Each cylinder has sixty-four 64-bit general purpose registers.

SRAM

Euterpe has 32KB of onchip SRAM dedicated to instructions and 32KB of onchip SRAM dedicated to data. Through Cerberus registers, each of these SRAMs may be configured as a mixture of cache and buffer. The cache is direct-mapped, virtually indexed, and physically tagged; it may be configured as 0KB, 4KB, 8KB, or 16KB in size, with the rest of the 32KB SRAM being buffer. Loads from data SRAM have a fixed latency of 2 cycles.

Cache

Offchip memory may be accessed indirectly, via the cache (cache misses are serviced by hardware), or directly using a *no-allocate* access mode associated with the address translation hardware. When offchip memory is accessed directly, loads are non-blocking and have variable latency, which depends on the speed of the memory and the contention for resources involved in accessing it. For more information on non-blocking loads, see "Non-blocking Load Buffer" on page 44.

Protection

Euterpe has a sophisticated address translation and protection mechanism. The protection mechanism involves 4 privilege levels: 0, 1, 2 and 3, where 0 is the lowest privilege and 3 is the highest. For more information on protection and memory management, see "Memory Hierarchy" on page 31.

MU 0054493

Gateways

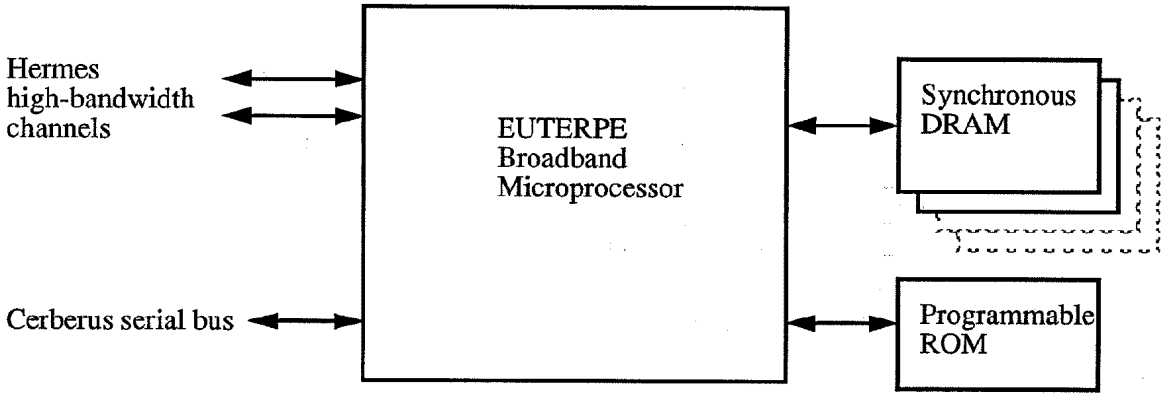
Euterpe provides *gateways* to control transitions between privilege levels. A gateway is a memory region that defines values for the *program counter* (pc), privilege level, and data pointer registers. A special branch instruction accesses a gateway; this form of access is separately permitted or denied based on protections established in the GTLB. Thus, what is essentially a function call can increase the privilege level in a controlled way. A special form of branch instruction is also provided to lower the privilege level upon return from the privileged function.

Input/Output

Highly Confidential

Euterpe provides external interfaces to between 4MB and 32MB of synchronous DRAM and up to 128KB of programmable ROM, as well as interfaces to the Cerberus serial bus (as master and slave), and to two Hermes high-bandwidth channels.

Euterpe Block Diagram



See page 32 for more information on these external interfaces.

MU 0054494

Chapter 2

Instruction Set

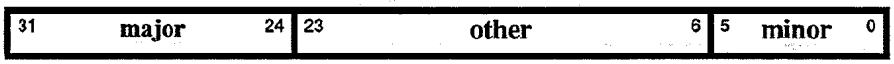
The tables on the following pages list the instructions that can be issued to the Euterpe processor. All instructions are 32 bits in size, and use the high-order 8 bits to specify a major operation code.

Operation code fields:

Byte order in memory:

byte 3	byte 2	byte 1	byte 0

For the major operation field values G.1, G.2, G.4, G.8, G.16, G.32, G.64, G.128, L.MINOR, S.MINOR. E.MINOR, and B.MINOR, the lowest-order six bits in the instruction specify a minor operation code.



Tables of Operation Codes

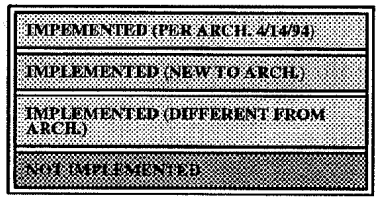
MU 0054496

MAJOR OPERATION CODE FIELD VALUES

MAJOR	0x0	20	40	60	80	A0	C0	E0
0x0	ERES	GSHUFFLEI	GFMULADD16	GFMULADD16	LU16LAI	SAAS64LAI	EADDIO	BFE16
1	ESHUFFLEI4MUX	GSHUFFLEI4MUX	GFMULADD32	GMULADD32	LU16BAI	SAAS64BAI	EADDIUO	BFNRE16
2		GSELECT8	GFMULADD64	GMULADD4	LU16LI	SCAS64LAI	ESETIL	BFNRGE16
3	EMDEPI	GMDEPI		GMULADD8	LU16BI	SCAS64BAI	ESETIGE	BFNUE16
4	EMUX	GMUX	GFMULSUB16	GMULADD16	LU32LAI	SMAS64LAI	ESETIE	BFE32
5	E8MUX	G8MUX	GFMULSUB32	GMULADD32	LU32BAI	SMAS64BAI	ESETINE	BFNUE32
6	EGFMUL64	GGFMUL8	GFMULSUB64	GMULADD64	LU32LI	SMUX64LAI	ESETIUL	BFNRGE32
7	ETRANSPOSE8MUX	GTRANSPOSE8MUX		GEXTRACT128	LU32BI	SMUX64BAI	ESETIUGE	BFNUE32
8					L16LAI	S16LAI	ESUBIO	BFE64
9	ECOPYSWAP	GCOPYSWAP		GFMULADD8	L16BAI	S16BAI	ESUBIUO	BFNUE64
A		GCOPYSWAPCOPY		GUMULADD4	L16LI	S16LI	ESUBIL	BFNUF3264
B		GCOPYSWAPSWAP		GUMULADD8	L16BI	S16BI	ESUBIGE	BFNUE64
C	EDEPI	GDEPI	GF16	GUMULADD16	L32LAI	S32LAI	ESUBIE	BFE128
D	EUDEPI	GUDEPI	GF32	GUMULADD32	L32BAI	S32BAI	ESUBINE	BFNUE128
E	EWTHI	GWTHI	GF64	GUMULADD64	L32LI	S32LI	ESUBIUL	BFNUGE128
F	EUWTHI	GUWTHI		GUEXTRACT128	L32BI	S32BI	ESUBIUGE	BFNUE128
10			GFMULADD16	GEXTRACTI	L64LAI	S64LAI	EADDI	BANDE
11			GFMULADD32	GEXTRACTI16 GUEXTRACTI32	L64BAI	S64BAI	EXORI	BANDNE
12			GFMULADD64	GEXTRACTI32	L64LI	S64LI	EORI	BL[a] BEZ
13			GFMULADD128	GUEXTRACTI64	L64BI	S64BI	EANDI	BGE[a] BGEZ
14			GFMULSUB16	GEXTRACTI64	L128LAI	S128LAI	ESUBI	BE
15			GFMULSUB32		L128LBI	S128BAI		BNE
16			GFMULSUB64	GEXTRACTI128	L128LI	S128LI	ENORI	BUL[a] BGZ
17			GFMULSUB128		L128BI	S128BI	ENANDI	BUGE[a] BLEZ
18				G.1	L8I	S8I		BGATEI
19				G.2	LUBI			
1A				G.4				
1B				G.8				
1C			GF16	G.16			ECOPYI	BI
1D			GF32	G.32			ENOOP	BLINKI
1E			GF64	G.64				
1F			GF128	G.128	L.MINOR	S.MINOR	E.MINOR	B.MINOR

a. The first instruction should be used when the registers are different. If the registers are equal, the second instruction ending with z should be used.

Highly Confidential



Most of the instructions in the major operation code table behave in a similar fashion to those described in the Terpsichore Architecture. The extract instructions in column 96, though, are somewhat different. For more information, see "Extract" on page 16.

MINOR OPERATION CODE FIELD VALUES

L.MINOR	0x0	8	10	18	20	28	30	38
0x0	LU16LA	L16LA	L64LA	L8				
1	LU16BA	L16BA	L64BA	LU8				
2	LU16L	L16L	L64L					
3	LU16B	L16B	L64B					
4	LU32LA	L32LA	L128LA					
5	LU32BA	L32BA	L128BA					
6	LU32L	L32L	L128L					
7	LU32B	L32B	L128B					

S.MINOR	0x0	8	10	18	20	28	30	38
0x0	SAAS64LA	S16LA	S64LA	S8				
1	SAAS64BA	S16BA	S64BA					
2	SCAS64LA	S16L	S64L					
3	SCAS64BA	S16B	S64B					
4	SMAS64LA	S32LA	S128LA					
5	SMAS64BA	S32BA	S128BA					
6	SMUX64LA	S32L	S128L					
7	SMUX64BA	S32B	S128B					

IMPEMENTED (PER ARCH 4/14/94)
IMPLEMENTED (NEW TO ARCH)
IMPLEMENTED (DIFFERENT FROM ARCH)
NOT IMPLEMENTED

MU 0054498

Highly Confidential

E.MINOR	0x0	8	10	18	20	28	30	38
0x0	EADDO	ESUBO	EANDN		EADD	ESUB	ESHLIO	ESHRI
1	EADDUO	ESUBUO	EXOR		ESHLO	ESHLUO		
2	ESETL	ESUBL	EOR				ESHLIUO	EUSHRI
3	ESEGE	ESUBGE	EAND		ELMS	EULMS		
4	ESETNE	ESUBE	EORN		EASUM	ESELECT8	ESHUFFLEI	EROTRI
5	ESETNE	ESUBNE	EXNOR		EROTL	ESHL		
6	ESETUL	ESUBUL	ENOR		ESHR	EUSHR	ESHLI	EMSHRI
7	ESETUGE	ESUBUGE	ENAND		EROTR	EMSHR		

B.MINOR	0x0	8	10	18	20	28	30	38
0x0	B							
1	BLINK							
2	BDOWN							
3								
4			BGATE	BBACK				
5								
6								
7								

IMPEMENTED (PER ARCH. 4/14/94)

IMPLEMENTED (NEW TO ARCH.)

IMPLEMENTED (DIFFERENT FROM ARCH.)

NOT IMPLEMENTED

MU 0054499

Highly Confidential

G.1	0x0	8	10	18	20	28	30	38
0x0		GMUL	GANDN				GEXPANDI	
1			GXOR		GCOMPRESS	GUCOMPRESS		
2			GOR				GUEXPANDI	
3			GAND					
4			GORN		GEXPAND	GUEXPAND	GCOMPRESSI	
5			GXNOR				GUCOMPRESSI	
6			GNOR					
7			GNAND					

G.2	0x0	8	10	18	20	28	30	38
0x0		GMUL			GADD	GSUB	GEXPANDI	GSHRI
1		GUMUL			GCOMPRESS	GUCOMPRESS		
2	GSETL						GUXPANDI	GUSHRI
3	GSETGE							
4	GSETE				GEXPAND	GUEXPAND	GCOMPRESSI	GROTRI
5	GSETNE				GROTL	GSHL	GUCOMPRESSI	
6	GSETUL				GSHR	GUSHR	GSHLI	GMSHRI
7	GSETUGE				GROTR	GMSHR		

G.4-32	0x0	8	10	18	20	28	30	38
0x0		GMUL			GADD	GSUB	GEXPANDI	GSHRI
1		GUMUL			GCOMPRESS	GUCOMPRESS		
2	GSETL						GUXPANDI	GUSHRI
3	GSETGE							
4	GSETE				GEXPAND	GUEXPAND	GCOMPRESSI	GROTRI
5	GSETNE				GROTL	GSHL	GUCOMPRESSI	
6	GSETUL				GSHR	GUSHR	GSHLI	GMSHRI
7	GSETUGE				GROTR	GMSHR		

IMPEMENTED (PER ARCH. 4/14/94)
IMPLEMENTED (NEW TO ARCH.)
IMPLEMENTED (DIFFERENT FROM ARCH.)
NOT IMPLEMENTED

MU 0054500

Highly Confidential

G.64	0x0	8	10	18	20	28	30	38
0x0		GMUL			GADD	GSUB	GEXPANDI	GSHRI
1		GUMUL			GCOMPRESS	GUCOMPRESS		
2	GSETL	GDIV					GUXPANDI	GUSHRI
3	GSETGE	GUDIV						
4	GSETE				GEXPAND	GUEXPAND	GCOMPRESSI [a]	GROTRI
5	GSETNE				GROTL	GSHL	GUCOMPRESSI [a]	
6	GSETUL				GSHR	GUSHR	GSHLI	GMSHRI
7	GSETUGE				GROTR	GMSHR		

a. The second half of GCOMPRESSI would be equivalent to ESHRI. The second half of GUCOMPRESSI would be equivalent to EUSHRI.

G.128	0x0	8	10	18	20	28	30	38
0x0								GSHRI
1								
2								GUSHRI
3								
4								GROTRI
5					GROTL	GSHL		
6					GSHR	GUSHR	GSHLI	GMSHRI
7					GROTR	GMSHR		

IMPEMENTED (PER ARCH, 4/14/94)
IMPLEMENTED (NEW TO ARCH)
IMPLEMENTED (DIFFERENT FROM ARCH)
NOT IMPLEMENTED

MU 0054501

XLU instruction encoding

This section describes the use of the instruction families in the XLU.

Shift/Rotate

Minor (E.MINOR)

Opcode column	Opcode row	Code	Operands		
32	5	EROTL	ra	rb	rc
32	7	EROTR	ra	rb	rc
40	5	ESHL	ra	rb	rc
32	1	ESHLO	ra	rb	rc
40	1	ESHLUO	ra	rb	rc
32	6	ESHR	ra	rb	rc
40	6	EUSHR	ra	rb	rc
40	7	EMSHR	ra	rb	rc

src1[63:0] = REG[ra][63:0]

src2[5:0] = REG[rb][5:0]

src3[63:0] = EMSHR ? REG[rc][63:0] : 0

dst1[63:0] = REG[rc][63:0]

use src2 as shift/rotate amount

use src3 for fill values (this will need to be replicated for the high-order copy of the result)

(If implemented, ESHLO and ESHLUO need support outside of the XLU.)

Opcode column	Opcode row	Code	Operands		
56	4	EROTRI	ra	rb	imm
48	6	ESHLI	ra	rb	imm
48	0	ESHLIO	ra	rb	imm
48	2	ESHLIUO	ra	rb	imm
56	0	ESHRI	ra	rb	imm
56	2	EUSHRI	ra	rb	imm
56	6	EMSHRI	ra	rb	imm

src1[63:0] = REG[ra][63:0]

src2[5:0] = imm[5:0]

src3[63:0] = EMSHRI ? REG[rb][63:0] : 0

dst1[63:0] = REG[rb][63:0]

use src2 as shift/rotate amount

use src3 for fill values (this will need to be replicated for the high-order copy of the result)

For EROTLI(x), use EROTRI((64 - x) & (64 - 1)).

(If implemented, ESHLIO and ESHLIUO need support outside of the XLU.)

Minor (G.size)

Opcode column	Opcode row	Code	Operands			Group Sizes
32	5	GROTL	ra	rb	rc	2..128
32	7	GROTR	ra	rb	rc	2..128
40	5	GSHL	ra	rb	rc	2..128
32	6	GSHR	ra	rb	rc	2..128
40	6	GUSHR	ra	rb	rc	2..128
40	7	GMSHR	ra	rb	rc	2..128

src1[127:0] = REG[ra+1][63:0] | REG[ra][63:0]

src2[6:0] = REG[rb][6:0]

src3[127:0] = GMSHR ? (REG[rc+1][63:0] | REG[rc][63:0]) : 0

dst1[127:0] = REG[rc+1][63:0] | REG[rc][63:0]

use (src2 & (size - 1)) as shift/rotate amount

use src3 for fill values

Generate a reserved instruction exception for G.1.

Opcode column	Opcode row	Code	Operands			Group Sizes
56	4, 5	GROTRI	ra	rb	imm	2..128
48	6, 7	GSHLI	ra	rb	imm	2..128
56	0, 1	GSHRI	ra	rb	imm	2..128
56	2, 3	GUSHRI	ra	rb	imm	2..128
56	6, 7	GMSHRI	ra	rb	imm	2..128

src1[127:0] = REG[ra+1][63:0] | REG[ra][63:0]

src2[6:0] = opminor[0] | imm[5:0]

src3[127:0] = GMSHRI ? (REG[rb+1][63:0] | REG[rb][63:0]) : 0

dst1[127:0] = REG[rb+1][63:0] | REG[rb][63:0]

use (src2 & (size - 1)) as shift/rotate amount

use src3 for fill values

MU 0054504

Generate a reserved instruction exception for G.1.

Generate a reserved instruction exception if opminor is odd and major is not G.128.

For GROTLI(x), use GROTRI((size - x) & (size - 1)).

Expand

Minor (G.size))

Opcode column	Opcode row	Code	Operands			Group Sizes
32	4	GEXPAND	ra	rb	rc	1..64
40	4	GUEXPAND	ra	rb	rc	1..64

src1[63:0] = REG[ra][63:0]

src2[6:0] = REG[rb][6:0]

dst1[127:0] = REG[rc+1][63:0] | REG[rc][63:0]

use (src2 & (2 * size - 1)) as shift amount

Generate a reserved instruction exception for G.128.

Opcode column	Opcode row	Code	Operands			Group Sizes
48	0, 1	GEXPANDI	ra	rb	imm	1..64
48	2, 3	GUEXPANDI	ra	rb	imm	1..64

src1[63:0] = REG[ra][63:0]

src2[6:0] = opminor[0] | imm[5:0]

dst1[127:0] = REG[rb+1][63:0] | REG[rb][63:0]

use (src2 & (2 * size - 1)) as shift amount

Generate a reserved instruction exception for G.128.

Generate a reserved instruction exception if opminor is odd and major is not G.64.

MU 0054505

Compress

Minor (G.size)

Opcode column	Opcode row	Code	Operands			Group Sizes
32	1	GCOMPRESS	ra	rb	rc	1..64
40	1	GUCOMPRESS	ra	rb	rc	1..64

src1[127:0] = REG[ra+1][63:0] | REG[ra][63:0]
src2[6:0] = REG[rb][6:0]
dst1[63:0] = REG[rc][63:0]
use (src2 & (2 * size - 1)) as shift amount
Generate a reserved instruction exception for G.128.

Opcode column	Opcode row	Code	Operands			Group Sizes
48	4	GCOMPRESSI	ra	rb	imm	1..64
48	5	GUCOMPRESSI	ra	rb	imm	1..64

src1[127:0] = REG[ra+1][63:0] | REG[ra][63:0]
src2[6:0] = 0 | imm[5:0]
dst1[63:0] = REG[rb][63:0]
use (src2 & (2 * size - 1)) as shift amount
Generate a reserved instruction exception for G.128.

Note:

GCOMPRESSI.64(ra, rb, 64+x) is equivalent to ESHRI(ra+1, rb, x)
GUCOMPRESSI.64(ra, rb, 64+x) is equivalent to EUSHRI(ra+1, rb, x)
Therefore, we avoid wasting opcodes on these cases.

MU 0054506

Extract

Major

Opcode column	Opcode row	Code	Operands			
96	7	GEXTRACT.128	ra	rb	rc	rd
96	15	GUEXTRACT.128	ra	rb	rc	rd

src1[255:0] = REG[ra+1][63:0] | REG[ra][63:0] |
REG[rb+1][63:0] | REG[rb][63:0]
src2[7:0] = REG[rc][7:0]
dst1[127:0] = REG[rd+1][63:0] | REG[rd][63:0]
use src2 as shift amount

Opcode column	Opcode row	Code	Operands			
96	16:23	GEXTRACTI.size	ra	rb	rc	imm
96	16:23	GUEXTRACTI.size	ra	rb	rc	imm

src1[255:0] = REG[ra+1][63:0] | REG[ra][63:0] |
REG[rb+1][63:0] | REG[rb][63:0]
src2[7:0] = (amount column of encoding table below)
size = (size column of encoding table below)
signed = (signed column of encoding table below)
dst1[127:0] = REG[rc+1][63:0] | REG[rc][63:0]
use (src2 & (2 * size - 1)) as shift amount

MU 0054507

Highly Confidential

Encoding Table

opcode	imm	signed	size	amount
96 23 *111	abcdef	s/u	128	01abcdef
96 22 *110	abcdef	s/u	128	00abcdef
96 21 *101	abcdef	s	64	x1abcdef
96 20 *100	abcdef	s/u	64	x0abcdef
96 19 *011	abcdef	u	64	x1abcdef
96 18 *010	1bcdef	s	32	xx1bcdef
96 18 *010	0bcdef	s/u	32	xx0bcdef
96 17 *001	1bcdef	u	32	xx1bcdef
96 17 *001	01cdef	s	16	xxx1cdef
96 17 *001	00cdef	s/u	16	xxx0cdef
96 16 *000	11cdef	u	16	xxx1cdef
96 16 *000	101def	s	8	xxxx1def
96 16 *000	100def	s/u	8	xxxx0def
96 16 *000	011def	u	8	xxxx1def
96 16 *000	0101ef	s	4	xxxxx1ef
96 16 *000	0100ef	s/u	4	xxxxx0ef
96 16 *000	0011ef	u	4	xxxxx1ef
96 16 *000	00101f	s	2	xxxxxx1f
96 16 *000	00100f	s/u	2	xxxxxx0f
96 16 *000	00011f	u	2	xxxxxx1f
96 16 *000	000101	s	1	xxxxxxx1
96 16 *000	000100	s/u	1	xxxxxxx0
96 16 *000	000011	u	1	xxxxxxx1
96 16 *000	000010	(trap)		
96 16 *000	000001	(trap)		
96 16 *000	000000	(trap)		

MU 0054508

Note that gextract(size) and guextract(size) are equivalent and are redundantly encoded for 1 <= size <= 64.

Note:

GEXTRACTI.128(ra, rb, rc, 128+x) is equivalent to GSHRI.128(ra, rc, x)
GUEXTRACTI.128(ra, rb, rc, 128+x) is equivalent to GUSHRI.128(ra, rc, x)

Therefore, we avoid wasting opcodes on these immediate cases.

Copy/Swap

Major

Opcode column	Opcode row	Code	Operands			
0	9	ECOPYSWAPI	ra	rb	imm1	imm2

src1[63:0] = REG[ra][63:0]
src2[5:0] = imm1[5:0]
src3[5:0] = imm2[5:0]
dst1[63:0] = REG[rb][63:0]

dst1[i] := src1[(i & src2) xor src3]

Opcode column	Opcode row	Code	Operands			
32	10	GCOPYSWAPI.COPY	ra	rb	imm1	imm2
32	9	GCOPYSWAPI	ra	rb	imm1	imm2
32	11	GCOPYSWAPI.SWAP	ra	rb	imm1	imm2

For GCOPYSWAPI.COPY:

src1[127:0] = x[63:0] | REG[ra][63:0]
src2[6:0] = 0 | imm1[5:0]
src3[6:0] = 0 | imm2[5:0]
dst1[127:0] = REG[rb+1][63:0] | REG[rb][63:0]

x[63:0] is 64 bits of *don't care*. Note that **ra** is used as a 64-bit operand, and may be even or odd.

MU 0054509

For GCOPYSWAPI:

src1[127:0] = REG[ra+1][63:0] | REG[ra][63:0]
src2[6:0] = 1 | imm1[5:0]
src3[6:0] = 0 | imm2[5:0]
dst1[127:0] = REG[rb+1][63:0] | REG[rb][63:0]

For GCOPYSWAPI.SWAP:

src1[127:0] = REG[ra+1][63:0] | REG[ra][63:0]
src2[6:0] = 1 | imm1[5:0]
src3[6:0] = 1 | imm2[5:0]
dst1[127:0] = REG[rb+1][63:0] | REG[rb][63:0]
dst1[i] := src1[(i & src2) xor src3]

The cases where the high or low 64-bits are replicated are both encoded as GCOPYSWAPI.COPY, where ra is an arbitrary 64-bit operand.

Note: For a pure 128-bit architecture, we would need four major opcodes to encode all of the combinations.

Transpose

Major

Opcode column	Opcode row	Code	Operands			
32	0	GSHUFFLEI.a.b.c	ra	rb	rc	imm
32	0	GSHUFFLEI.id	ra	rb	rc	imm=0

src1[127:0] = REG[ra][63:0] | REG[rb][63:0]
src2[5:0] = imm[5:0]
dst1[127:0] = REG[rc+1][63:0] | REG[rc][63:0]

$a = 2^x, b = 2^y, c = 2^z$

$a <= 128, b >= 1, c >= 2$
$c < a/b$

$x <= 7, y >= 0, z >= 1$
$z < x-y$

MU 0054510

src2 = id ? 0
: $(x^3 - 3*x^2 - 4*x)/6 - (z^2 - z)/2 + x*z + y + 1$
src2 = 0 is a nop (useless here, but needed for shuffleimux)
1 <= src2 <= 56 is a shuffle/deal
src2 >= 57 causes a reserved instruction exception

dst1[i] := src1[src2 == 0 ? i : iperm(x, y, z, i)]

where iperm(x, y, z, i) is
(i & (((1 << (7 - x)) - 1) << x)) |
((i << (x - z - y)) & (((1 << z) - 1) << (x - z))) |
((i >> z) & (((1 << (x - z - y)) - 1) << y)) |
(i & ((1 << y) - 1))
i.e., preserve 7-x high bits, preserve y low bits, and rotate remaining middle bits right by z.

Minor (E.MINOR)

Opcode column	Opcode row	Code	Operands		
48	4	ESHUFFLEI.a.b.c	ra	rb	imm
48	4	ESHUFFLEI.id	ra	rb	imm=0

src1[63:0] = REG[ra][63:0]
src2[5:0] = imm[5:0]
dst1[63:0] = REG[rb][63:0]

Interpret control as above, but require a <= 64, which is the same as control <= 35. Control >= 36 causes a reserved instruction exception.
dst1[i] := src1[src2 == 0 ? i : iperm(x, y, z, i)]

Shuffle/mux

Major

Opcode column	Opcode row	Code	Operands			
32	7	GTRANSPOSE8MUX	ra	rb	rc	rd
32	5	G8MUX	ra	rb	rc	rd

src1[127:0] = REG[ra+1][63:0] | REG[ra][63:0]
src2[5:0] = G8MUX ? 0 : 30
src3[191:0] = REG[rc][63:0] | REG[rb+1][63:0] | REG[rb][63:0]
dst1[127:0] = REG[rd+1][63:0] | REG[rd][63:0]

Use src2 for shuffle control.
Use src3 for mux control.
Replicate mux control for high and low 64 bits.

Note that GTRANSPOSE8MUX is equivalent to GSHUFFLEI8MUX.64.1.8.
Note that G8MUX is equivalent to GSHUFFLEI8MUX.id.

use tmp[127:0] as follows:

```
tmp[i] := src1[src2 == 0 ? i : iperm(6, 0, 3, i)]
dst1[i] := tmp[(i & 0x78) |
            (src3 [(i & 0x3f) + 128] << 2) |
            ( src3[(i & 0x3f) + 64] << 1) |
            src3[i & 0x3f]]
```

Opcode column	Opcode row	Code	Operands			
32	1	GSHUFFLEI4MUX.a.b.c	ra	rb	rc	imm
32	1	G4MUX	ra	rb	rc	imm=0

src1[127:0] = REG[ra+1][63:0] | REG[ra][63:0]
src2[5:0] = imm[5:0]
src3[127:0] = REG[rb+1][63:0] | REG[rb][63:0]
dst1[127:0] = REG[rc+1][63:0] | REG[rc][63:0] **MU 0054512**

Use src2 for shuffle control.
Use src3 for mux control.
Replicate mux control for high and low 64 bits.
Shuffle control >= 57 causes a reserved instruction exception.

use tmp[127:0] as follows:

```
tmp[i] := src1[src2 == 0 ? i : iperm(x, y, z, i)]
dst1[i] := tmp[(i & 0x7c) |
            (src3[(i & 0x3f) + 64] << 1) |
            src3[i & 0x3f]]
```

Opcode column	Opcode row	Code	Operands			
0	7	ETRANSPOSE8MUX	ra	rb	rc	rd
0	5	E8MUX	ra	rb	rc	rd

src1[63:0] = REG[ra][63:0]
src2[5:0] = E8MUX ? 0 : 30
src3[191:0] = REG[rc][63:0] | REG[rb+1][63:0] | REG[rb][63:0]
dst1[63:0] = REG[rd][63:0]

Use src2 for shuffle control.

Use src3 for mux control.

Note that ETRANSPOSE8MUX is equivalent to ESHUFFLEI8MUX.64.1.8.

Note that E8MUX is equivalent to ESHUFFLEI8MUX.id.

use tmp[63:0] as follows:

```
tmp[i] := src1[src2 == 0 ? i : iperm(6, 0, 3, i)]
dst1[i] := tmp[(i & 0x38) |
            (src3[i + 128] << 2) |
            (src3[i + 64] << 1) |
            src3[i]]
```

Opcode column	Opcode row	Code	Operands			
0	1	ESHUFFLEI4MUX.a.b.c	ra	rb	rc	imm
0	1	E4MUX	ra	rb	rc	imm=0

src1[63:0] = REG[ra][63:0]
src2[5:0] = imm[5:0]
src3[127:0] = REG[rb+1][63:0] | REG[rb][63:0]
dst1[63:0] = REG[rc][63:0]

Use src2 for shuffle control.

MU 0054513

Highly Confidential

Use src3 for mux control.

Shuffle control >= 36 causes a reserved instruction exception.

use tmp[63:0] as follows:

```
tmp[i] := src1[src2 == 0 ? i : iperm(x, y, z, i)]
dst1[i] := tmp[(i & 0x3c) |
            (src3[i + 64] << 1) |
            src3[i]]
```

Select

Major

Opcode column	Opcode row	Code	Operands
32	2	GSELECT8	ra rb rc rd

src1[127:0] = REG[ra][63:0] | REG[rb][63:0]

src2[63:0] = REG[rc][63:0]

dst1[127:0] = REG[rd+1][63:0] | REG[rd][63:0]

dst1[i] := src1[(src2[(i/8) * 4 + 3 : (i/8) * 4] * 8) + (i%8)]

Minor (E.MINOR)

Opcode column	Opcode row	Code	Operands
40	4	ESELECT8	ra rb rc

src1[63:0] = REG[ra][63:0]

src2[31:0] = REG[rb][31:0]

dst1[63:0] = REG[rc][63:0]

dst1[i] := src1[(src2[(i/8) * 4 + 2 : (i/8) * 4] * 8) + (i%8)]

MU 0054514

Bit Field Withdraw/Deposit

Major

Opcode column	Opcode row	Code	Operands			
32	12	GDEPI	ra	rb	imm1	imm2
32	13	GUDEPI	ra	rb	imm1	imm2
32	14	GWTHI	ra	rb	imm1	imm2
32	15	GUWTHI	ra	rb	imm1	imm2
32	3	GMDEPI	ra	rb	imm1	imm2

src1[127:0] = REG[ra+1][63:0] | REG[ra][63:0]
size = (size column of encoding table below)
src2[5:0] = imm1[5:0]
src3[5:0] = imm2[5:0]
src4[127:0] = GMDEPI ? (REG[rb+1][63:0] | REG[rb][63:0]) : 0
dst1[127:0] = REG[rb+1][63:0] | REG[rb][63:0]
use (src2 & (size - 1)) as shift amount
use (src3 & (size - 1)) as fsize - 1
use src4 for fill values

Opcode column	Opcode row	Code	Operands			
0	12	EDEPI	ra	rb	imm1	imm2
0	13	EUDEPI	ra	rb	imm1	imm2
0	14	EWTHI	ra	rb	imm1	imm2
0	15	EUWTHI	ra	rb	imm1	imm2
0	3	EMDEPI	ra	rb	imm1	imm2

src1[63:0] = REG[ra][63:0]
size = (size column of encoding tablebelow)
src2[5:0] = imm1[5:0]
src3[5:0] = imm2[5:0]
src4[63:0] = EMDEPI ? REG[rb][63:0] : 0
dst1[63:0] = REG[rb][63:0]
use (src2 & (size - 1)) as shift amount
use (src3 & (size - 1)) as fsize - 1

MU 0054515

Highly Confidential

use src4 for fill values (this will need to be replicated for the high-order copy of the result)

Encoding Table

size	amount	fsize-1	imm1	imm2
64	abcdef	pqrstu	abcdef	pqrstu
32	0bcdef	0qrstu	1bcdef	1qrstu
16	00cdef	00rstu	11cdef	11rstu
8	000def	000stu	111def	111stu
4	0000ef	0000tu	1111ef	1111tu
2	00000f	00000u	11111f	11111u
1	000000	000000	111111	111111

Range restriction:

0 <= amount < size

1 <= fsize <= size - amount

The group size can be obtained from imm1 & imm2 as follows:

imm1 & imm2	size
0xxxxx	64
10xxxx	32
110xxx	16
1110xx	8
11110x	4
111110	2
111111	1

MU 0054516

Illegal combinations occur when **amount + fsize - 1 >= size**, or equivalently, **(imm1 & (size - 1)) + (imm2 & (size - 1)) >= size**, in which case a reserved instruction exception is generated.

The only group size which is allowed for the *E* instructions is 64. So, for the *E* instructions, an exception is generated if the high bit of **imm1 &**

imm2 is 1. This is in addition to the other illegal combinations noted above.

Group sizes 1 through 64 are allowed for the *G* instructions.

Note that all valid combinations in which **imm2 = 111111** are nops, since **fsize - 1** will be **size - 1** so fsize will equal size and amount will have to be zero (so imm1 will be **64 - size**). The one valid combination for a group size of 1 is merely another instance of this, so it isn't treated specially here. If we want to disallow this entire class, the test (**imm2 = 111111**) is trivial.

Notes:

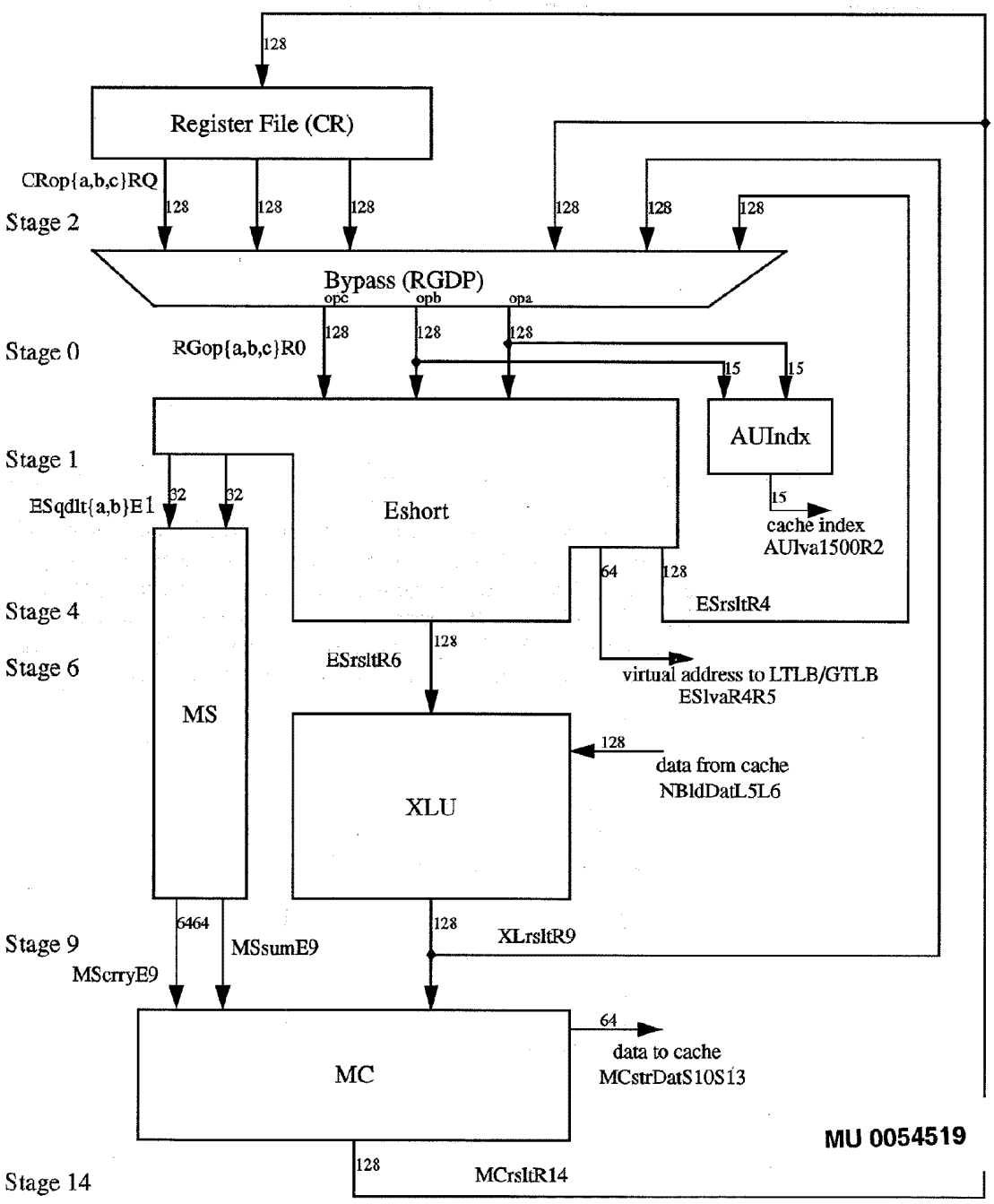
- DEP is equivalent to group left shift by (size - fsize) followed by a group signed right shift by (size - fsize - amount).
- UDEP is equivalent to group left shift by (size - fsize) followed by a group unsigned right shift by (size - fsize - amount).
- MDEP is equivalent to group left shift by (size - fsize) followed by a group unsigned right shift by (size - fsize - amount). However, only fsize bits of each result are changed (i.e., those bits taken from **ra**). The remaining (size - fsize) bits of each result are preserved.
- WTH is equivalent to group left shift by (size - fsize - amount) followed by a group signed right shift by (size - fsize).
- UWTH is equivalent to group left shift by (size - fsize - amount) followed by a group unsigned right shift by (size - fsize).

Chapter 3

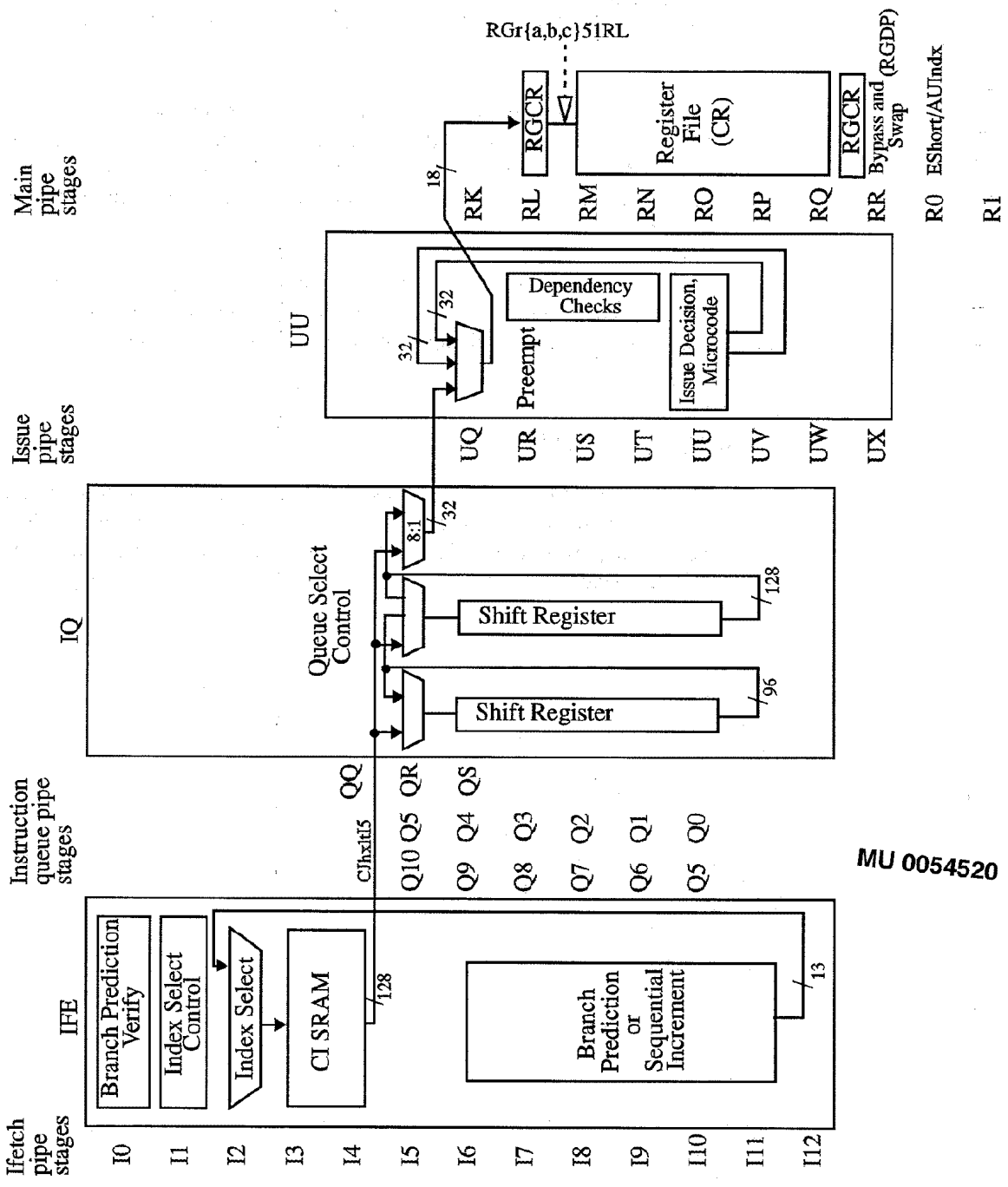
Pipeline Organization

The Terpsichore architecture performs all instructions in a given cylinder as if executed one-by-one, in order, with precise exceptions always available. Consequently, code which ignores the Euterpe pipeline implementation will still perform correctly. However, the highest performance of the Euterpe processor is achieved only by matching the ordering of instructions to the characteristics of the pipeline.

Main Pipeline



Instruction Pipeline



Branch Prediction

The branch predictor guesses the next instruction cache index based on the outcome of the last time this branch was encountered by the cylinder. The definition of *this branch* is based on the cache index of the branch instruction.

There is only one branch prediction register per cylinder. Only immediate backwards branches are predicted. If there is no entry in the prediction register for this branch, either because the register was overwritten or because this is the first time that this branch is encountered, no prediction is attempted.

False prediction can happen if the next branch after a predicted branch has the same cache index as the predicted branch. This might occur if the instruction stream has changed or through cache index aliasing.

PC Catchup Condition

Any time a predicted branch is taken, calculations of the current program count (pc) fall behind. This condition is allowed to persist until a full pc is needed, say by a branch. To catch up, the instruction fetch pipeline stalls for one cycle.

Chapter 4

Memory Hierarchy

This chapter discusses the memory interfaces, translation mechanisms, caches, and non-blocking load buffer of the Euterpe microarchitecture.

The facilities of the memory management system are themselves memory mapped, in order to provide for the manipulation of these facilities by high-level language, compiled code. See page 32 for a complete list. The table also shows whether the data access type is *Fast on-chip*, or through the *non-blocking load buffer* (NB). For more information on the NB, see page 44.

In MicroUnity's system software, privileged software runs at level 2 or 3. Virtual addresses are 64 bits wide.

The translation mechanism allows full fine-grained (down to 64 byte) control of access to the virtual address space. For information on address translation, see page 36.

Physical Address Space

Region	Space	Address	Data Access Type	Executable Code Space
Non-interleaved Hermes (0-7)	0x0	0x000000000000		
Channel 0 Module 0		0x000000000000	NB	X [a]
Channel 0 Module 1		0x000800000000	NB	X [a]
Channel 1 Module 0		0x002000000000	NB	X [a]
Channel 1 Module 3		0x003800000000	NB	X [a]
Euterpe DRAM	0x20	0x200000000000	NB	X
Euterpe ROM (Octlet Access)	0x40	0x400000000000	NB	X
Euterpe ROM (Byte Access)	0x50	0x500000000000	NB	
Cerberus (Serial Bus)	0x60-0x67	0x600000000000		
Net 0 Node 0		0x600000000000	NB	X [a]
Net 0 Node 1		0x600000080000	NB	X [a]
Net 0 Node 255		0x600007f80000	NB	X [a]
Net 1 Node 0		0x600008000000	NB	X [a]
Net 65535 Node 255		0x67ffff80000	NB	X [a]
On-chip Space	0x80	0x800000000000		
Global TLB		0x8000007f0000	NB	
Local TLB		0x8000007f8000	NB	
Daemon Access		0x8000007ffd80	NB	
Keypad/Display		0x8000007ffdc0	NB	
Current cylinder		0x8000007ffe00	Fast [b]	
Cycle count		0x8000007ffe40	Fast [b]	
Timer match		0x8000007ffe80	Fast [b]	
Watchdog match		0x8000007ffec0	Fast [b]	
Event register		0x8000007fff00	Fast [b]	
Event mask		0x8000007fff40	Fast [b]	
Exception Address		0x8000007fff80	Fast [b]	
Exception Status		0x8000007fffc0	Fast [b]	
Data Buffer		0x800000800000	Fast [b]	
Data Cache		0x800000804000	Fast [b]	
Data Cache Tags		0x800000a00000	NB	
Instruction Buffer		0x800000c00000	NB [c]	X
Instruction Cache		0x800000c04000	NB [c]	X
Instruction Cache Tags		0x800000e00000	NB	

MU 0054523

a. Executable if devices are present.
b. Fast on-chip access, not through NB load buffer. 2 major cycle Load latency; 4 major cycle Store latency.
c. Instruction fetch is Fast.

Unused Address Space

The following table shows the results of accessing portions of the assigned address space that are unused.

Unused Space (Addr 47:0)	Result
Hermes (0x000000000000..0x1fffffffffff) - The amount of unused space depends on which channels are enabled.	
Implemented device (0x000000000000..0x003fffffffff)	If the device doesn't exist, machine check. Otherwise, the result will be device-specific (within modules space) either returning 0 or machine check.
Unimplemented device (0x004000000000..0x1fffffffffff)	machine check
DRAM - The amount of space depends on the configuration set in Cerberus register 10	*wrapping - no unused space*
ROM (Octlet Access)	*wrapping - no unused space*
ROM (Byte Access)	*wrapping - no unused space*
Cerberus (0x6000000000..0x6700000000000)	If the device doesn't exist, machine check. Otherwise, the result will be device-specific (within devices space) either returning 0 or machine check.
unused (0x6800000000000..0x7fffffffffff)	exception 11 (see page 41)
On-chip Space (0x800000000000 + offset below)	
Global TLB (0x000000..0x7effff)	exception 11 (see page 41)
Global TLB (0x7f0800..0x7f7fff)	exception 11 (see page 41)
Local TLB (0x7f8008..0x7f803f)	returns 0
Local TLB (0x7f8040..0x7ffd7f)	exception 11 (see page 41)
Daemon Access (0x7ffd88..0x7ffdbf)	returns 0
Keyboard/Display (0x7ffdc8..0x7ffdff)	returns 0
Current cylinder (0x7ffe08..0x7ffe3f)	returns 0
Cycle count (0x7ffe48..0x7ffe7f)	returns 0
Timer match (0x7ffe88..0x7ffebf)	returns 0
Watchdog timer (0x77ffec8..0x7ffeff)	returns 0
Event register (0x7fff08..0x7fff0f)	returns 0
Event register (0x7fff18..0x7fff1f	returns 0
Event register (0x7fff28..0x7fff2f)	returns 0
Event register (0x7fff38..0x7fff3f)	returns 0
Event mask (0x7fff48..0x7fff7f)	returns 0
Exception Address (0x7fff88..0x7fffbf)	returns 0
Exception Status (0x7fffc8..0x7fffff)	returns 0
Data Buffer (0x808000..0x89ffff)	exception 11 (see page 41)
Data Tags	*repeats every 4K*
Instruction Buffer (0xc08000..0xdfffff)	exception 11 (see page 41)
Instruction Tags	*repeats every 4K*
unused (0x800001000000..0xffffffffffff)	exception 11 (see page 41)

MU 0054524

Hermes

MicroUnity's Hermes high-bandwidth channel architecture is designed to provide ultra-high bandwidth communications between devices within the Terpsichore architecture. Euterpe has two Hermes channels. They comply with the specifications of the Terpsichore architecture. Channels 2 through 7 are not used in this implementation.

Euterpe DRAM

Euterpe DRAM is a JEDEC standard synchronous DRAM interface. There are 32 bidirectional data pins. Euterpe can address up to 16 MB of DRAM. Some of the characteristics of the DRAM interface, including the size, are programmable through Cerberus register 10. See "Cerberus Registers" on page 60 for details on Cerberus register fields and settings.

Euterpe ROM

Euterpe ROM is electronically programmable. It has two areas which can be accessed by octlet reads. In the byte-access area, only the low-order byte of the octlet is read. The process for writing the EE prompt is vendor-specific, so it is the responsibility of software to define the correct process.

The processor boots from a starting address in ROM. Euterpe can address up to 128KB of ROM.

Cerberus

Euterpe's implementation of the Cerberus serial bus is described in "Cerberus Registers" on page 58.

MU 0054525

On-chip Space

On-chip Space includes the GTLB, LTLB, system registers, Data Buffer,

Cache, and Tags, and Instruction Buffer, Cache, and Tags.

Euterpe's implementation details for the on-chip area are described below. Note that On-chip Space cannot be cached.

GTLB

The global translation lookaside buffer (GTLB) address given in the table on page 32 is the base physical address for the GTLB entries. There are 64 shared entries, with entry **0** being assigned to the base GTLB address.

Bits 10:5 of the physical address are the logical address of each GTLB entry. Each entry has four fields in it, addressed by bits 4:3 of the physical address. Bits 2:0 of the physical address are ignored.

Each field must be read or written as an individual 64-bit field. The significant bits of the **protection** field are 15:0. The significant bits of the **xor**, **match**, and **mask** fields are 63:6.

offset	Entry 0..63															
0x00	63												16	15	protection	0
0x08	63	xor											6	5		0
0x10	63	match											6	5		0
0x18	63	mask											6	5		0

Each entry's starting address is offset 0x20 from the start of the previous entry. For example, entry 1 has an offset of 0x20 from the base GTLB address, entry 2 has an offset of 0x40, and so on. The offset of each field shown in the diagram is added to the starting physical address for each entry.

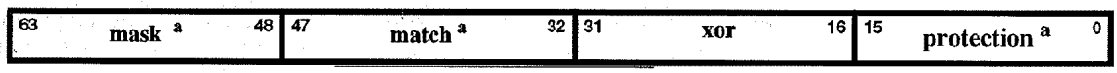
GTLB Protection Field Format

bit	Meaning in HW
15	high priority if set, low if not
14..13	Caching control 00 ignored 01 exception 7 10 force non-blocking ld/st if cache miss 11 force non-blocking ld/st
12	Detail access, exception 6 if set
11..8	ignored by hardware
7..6	Read access, causes exception 5 if > privilege level
5..4	Write access, causes exception 5 if > privilege level
3..2	Execute access, causes exception 5 if > privilege level
1..0	Gateway access, causes exception 5 if > privilege level

See page 41 for more details on exceptions.

LTLB

There is one local translation lookaside buffer (LTLB) for each cylinder. The xor field is used for address translation. The mask, match, and protection fields of each LTLB are unused. Writing a non-zero value to them has no effect.



a. In Euterpe, the mask field is always 0xffff, and the match and protection fields are always 0x0000.

MU 0054527

Address Translation

There are three levels of addresses in Euterpe: local virtual (LVA), global virtual (GVA), and the physical address (PA). User code uses LVA's. LVA's are translated to GVA's using the data in the LTLB to translate the upper 16 bits of the LVA.

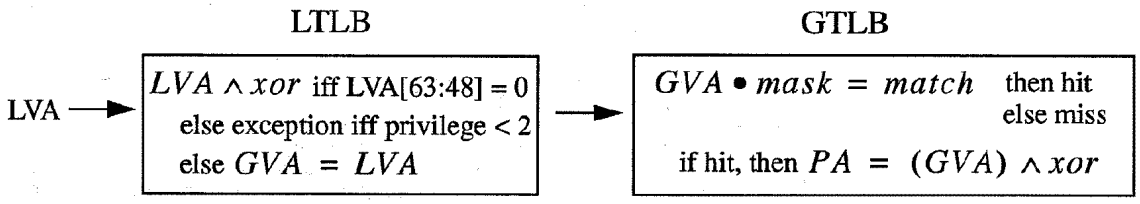
The LTLB can remap memory segments of 2^{48} bytes or larger. There is one LTLB for each cylinder, which allows all five cylinders to execute in the same memory segment, and be mapped to the same, or different, GVA's.

GVA's are directly accessible to privileged system software through the

default identity translation. The GVA can be translated to a PA using the data in the GTLB to translate the upper 58 bits of the GVA.

The GTLB entries can remap memory segments of 64 bytes (a cache line) or larger.

With the following formula, you can determine a physical address from the virtual address using the data from the LTLB and GTLB.



This design allows for page sizes to be any power of 2, beginning at a page size of 64 bytes. Good use is made of this in the MicroUnity system software.

If multiple hits occur in the GTLB, the results are unpredictable.

System Registers

The system registers provide access to vital information and control for Event Handling (see page 51), Keypad and Display, and the Watchdog Timer.

In general, accesses to system registers take the same time as accesses to the cache, with loads from system registers having a fixed latency of 2.

The contents of system registers are undefined on reset. For more information on reset, see "Reset and Error Recovery" on page 53

Daemon Access Register

MU 0054528

This 64 bit register is shared by all cylinders.

Storing the memory mapped physical address of an I/O device's Event register to the Daemon access register causes Euterpe to issue a blocking read of the I/O device's register. The blocking read returns when the register is non-zero. Euterpe will automatically *OR* the contents of the I/O device's Event register into its own Event register.

Keypad/Display

This 64 bit register is shared by all cylinders.

The LED/keypad interface register has been assigned physical address 8000007ffdc0. This register will have access characteristics similar to that of the flash ROM and GTLB, for example it is accessed via the non-blocking load buffer, but should be much faster than a Cerberus register.

The layout is as follows (where bit numbers refer to what you will get using a little endian load):

bits	function	note
63..48	keypad state	This field is read-only. Actual mapping of keys to bit position is controlled by pcb routing. Current plan on hestia is to implement only 12 of the 16 possible buttons. The decoder should correctly report two buttons pressed simultaneously.
47..43	reserved	
42..40	intensity	Controls display intensity by modulating driver pulsewidth.
39..32	colon/am/pm	mapping between bits and segments controlled by pcb, TBD.
31..24	10's Hrs	bits 0..7 of this byte control segments a : g respectively. See LED spec for details.
23..16	1's Hrs	bits 0..7 of this byte control segments a : g respectively. See LED spec for details.
15..8	10's Mins	bits 0..7 of this byte control segments a : g respectively. See LED spec for details.
7..0	1's Mins	bits 0..7 of this byte control segments a : g respectively. See LED spec for details.

Hardware provides the basic scanning function for both the display matrix and the keypad matrix. Software is responsible for polling the keypad state and debouncing the buttons. For the display, software must encode the required segment patterns and update the display register whenever the display is to change. If the colon needs to flash, it's a software responsibility. The intensity field controls the on/off ratio of the scan circuit over an 8 : 1 range. The display can be blanked completely by writing a segment pattern of all 0's.

MU 0054529

Current Cylinder

This 64 bit register is unique to each cylinder, and contains the cylinder number. This register is read-only. Writes are ignored.

Cycle Count Register

This 32 bit register is shared by all cylinders. The register is accessed as a 64 bit register, but the most significant 32 bits are unused and always return zero.

This register counts minor cycles (ticks).

Timer Match Register

This 32 bit register is shared by all cylinders. The register is accessed as a 64 bit register, but the most significant 32 bits are unused, and always return zero.

When the contents of the Cycle count register and the Timer match register are equal, bit 0 in the Event register is set. The bit remains set until it is cleared by software or a reset occurs.

Watchdog Match Register

This 32 bit register is shared by all cylinders. The register is accessed as a 64 bit register, but the most significant 32 bits are unused, and always return zero.

The Watchdog match register detects hangs in software or hardware. The value should continue to be reset to stay ahead of the cycle count. If the contents of the Cycle count register and the Watchdog match register are equal, it causes a machine check. See page 55 for more information on machine checks.

The register is disabled until a first value is written to it. If the register is not enabled, then code can run without this check.

Event Register

This 64 bit register is shared by all cylinders.

The Event register appears at four locations in the Event register field, with slightly different side-effects on read and write operations for each location. The offsets shown in the table below are from the Event register

address shown on page 32.

offset	side effect on read	side effect on write
0x00	none: return Event register contents	normal: sets or clears bits in the register contents by writing 1 or 0
0x10	none: return Event register contents	normal: sets or clears bits in the register contents by writing 1 or 0
0x20	reads as zero	*sets* bits in the register for bits set in the operand value
0x30	reads as zero	*clears* bits in the register for bits set in the operand value

The normal way to acknowledge events is with the bit clear above. This will leave any bits set which are not being explicitly acknowledged.

Neither blocking reads nor blocking writes are supported to the event register. Accesses to the blocking address, as defined in *Terpsichore System Architecture*, are treated exactly the same as accesses to the non-blocking address.

Event Mask Register

Each cylinder has a unique 64 bit Event mask register. Each of these registers appears at the same location in the physical address space. One cylinder, therefore, cannot access another cylinder's Event mask register.

The event mask register determines which interrupts a cylinder can receive. Each bit corresponds to a bit in the Event register.

If a cylinder sets a bit in the event register, for which there is no event mask bit uncovered, and later a second cylinder uncovers that bit, the second cylinder will be interrupted.

Exception Address Register

This 64 bit register is shared by all cylinders, and is locked by a cylinder until it returns from its event handler.

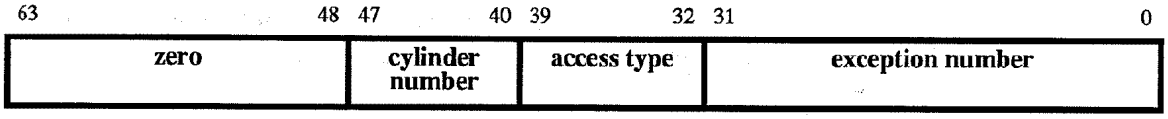
The Exception address register contains the Local Virtual Address (LVA) of the data or branch target of the instruction that caused the addressing exception. Exceptions 1 through 12 are addressing exceptions. See the table below for more information.

Exception Status Register

This 64 bit register is shared by all cylinders, and is locked when a cylinder takes an exception until it returns from its event handler.

This register contains the cylinder number, access type, and exception number.

The Exception Status register is read-only and has the format shown below. A table of the *exception number* interpretation follows the format diagram.



The exception numbers are one greater than those shown in the Terpsichore Architecture to make room for a *no exception* at number 0. This means that, if you are in an event handler, and the exception number is 0 here, then an interrupt put you in the event handler.

As well, since the LTLB is a subset LTLB, some of its exception numbers were not needed for this implementation, and those numbers are being used for different purposes. Numbers that have been replaced are marked with an asterisk (*) are described in more detail following the table.

number	exception
0	None
1	Access disallowed by tag
2	Access detail required by tag
3	Cache coherence action required by tag
4	Access disallowed by virtual address
5	Access disallowed by global TLB
6	Access detail required by global TLB
7	Cache coherence action required by global TLB
8	Global TLB miss
9 *	Base not equal to lva exception
10 *	Lva not equal to pa exception

MU 0054532

11 *	Illegal Physical Address
12	Local TLB miss
13	Floating-point arithmetic [a]
14	Fixed-point arithmetic
15	Reserved instruction

a. Reserved. Floating-point arithmetic is not currently implemented.

New Exceptions

Exception 9 occurs if LVA[63:47] does not equal Ra[63:47] when the priority level is 2 or 3. Ra is the base register.

Exception 10 occurs if:

- LVA[47] docs not cqual PA[47]
- LVA[47] equals 1, but LVA[15:6] does not equal PA[15:6].

Exception 11 occurs if:

- a program accesses one of the unused address spaces noted on page 33
- the cache is set to zero, but the GTLB is set to cache.

Multiple Exceptions

Exceptions are precise, so if multiple exception conditions occur, they must all be for the same instruction. Since only one exception can be reported at a time, multiple exceptions in multiple cylinders will serialize to usc the exception status register.

Buffers, Caches, and Tags

There are separate buffers, caches, and tags for data and instructions. Their characteristics are mostly the same. The differences are listed below.

Buffers and Caches

MU 0054533

The Data buffer and cache share the same physical space. Through Cerberus Register 6, the relative sizes of the buffer and cache can be varied, though the total size of the Data buffer and cache together is always 32KB. The same is true of the Instruction buffer and cache. The caches are separately programmable. The size of the Data and Instruction tags vary according to the size of Data and Instruction caches. The legal

configurations appear below.

Configuration	Buffer size	Cache size	D/ICache starting phsical address	D/ITag starting physical address	Cache tag index
0	32KB	0KB	n/a	n/a	n/a
12	28KB	4KB	0x800000807000/ 0x800000c07000	0x800000a00300/ 0x800000e00300	LVA[11:6]
13	24KB	8KB	0x800000806000/ 0x800000c06000	0x800000a00200/ 0x800000e00200	LVA[12:6]
14	16KB	16KB	0x800000804000/ 0x800000c04000	0x800000a00000/ 0x800000e00000	LVA[13:6]

The caches are memory-mapped and write-back. The cache line size is 64 bytes. The Data cache is virtually indexed and physically tagged, and the Instruction cache is virtually indexed and virtually tagged.

Data Cache Tag

The Data cache tag is 48 bits wide. Bits 47:6 of the Data cache tag are the physical tag. Bits 5:0 of the cache tag are the protection field. The data cahce tag is accessed as an octlet, though bits 63:48 are not implemented and will always return all zeros.

The following table shows the format of the Data cache tag protection field.

bit	Protection Field
5	Caching control, exception 3 if set
4	Detail access, exception 2 if set
3..1	ignored by hardware
0	used for dirty bit by hardware

Initial values of bits 5 and 4 are inherited from the GTLB, but software can write to them to change the values. The data cache tag cannot cause an exception 1 in this implementation.

MU 0054534

Instruction Cache Tag

The Instruction cache tag is 64 bits wide. Bits 63:12 of the Instruction cache tag are the global virtual tag. Bits 7:0 of the cache tag are the protection field. Bits 11:8 are unused.

The following table shows the format of the Instruction cache tag protection field.

bit	Protection Field
7	Caching control, exception 3 if set
6	Detail access, exception 2 if set
5..2	ignored by hardware
1..0	Execution access, causes exception 1 if > privilege level

Initially, the values of bits 7:2 are zero. The value of the Exception access field [1:0] is inherited from the GTLB, but software can write to it to change the value.

MU 0054535

Non-blocking Load Buffer

Queueing multiple memory requests is useful for maintaining a high throughput to and from the DRAM and Hermes channel devices without stalling the processor unnecessarily. Multiple memory requests can be issued to these devices, allowing overlap, or pipelining, of the latencies for a single request.

Requests can be sent until the non-blocking load buffer or device queues are full or until a register dependency is identified. If a register dependency occurs, the affected thread is stalled until the dependency is satisfied.

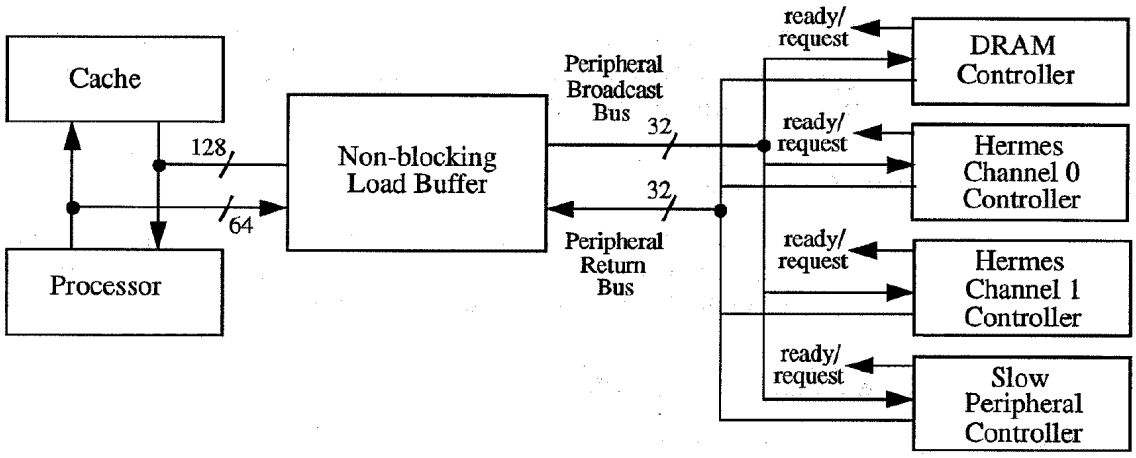
All non-blocking accesses go through the non-blocking load buffer. For incoherent (cached) accesses, the cache controller uses the non-blocking load buffer for fills and writebacks. For uncached physical accesses, each request causes an entry in the non-blocking load buffer. *No-allocate* accesses in the case of a cache miss also cause an entry in the non-blocking load buffer.

If the GTLB protection information indicates that the memory region is cached, a separate piece of logic, the cache controller, translates a cache miss into a cache fill (four non-blocking load buffer entries) and, if the dirty bit is set, a cache writeback (another four non-blocking load buffer entries).

The non-blocking load buffer consists of 16 entries. The non-blocking priority bit in the GTLB determines whether a request is low or high priority. If the number of valid entries exceeds the value in the NB Priority

field in Cerberus register 6, low priority reqests will be rejected and the cylinder will stall. All cache fills and writebacks are low priority. If memory management is disabled, all requests are low priority.

Non-blocking Load Buffer Connections



Cache and Processor

PBB and PRB

All non-blocking load entries are serviced by the Peripheral Broadcast Bus (PBB) and the Peripheral Response Bus (PRB). Requests are sent from the NB load buffer to the peripheral controllers on the PBB and received back on the PRB.

MU 0054536

Peripheral Controllers

There are five independent queues: two for DRAM, one for each Hermes channel, and one for all other requests. In the non-blocking control path, there is no cache miss queue.

The hardware has no special interlocks to prevent reordering if the software sets up multiple mappings onto the same physical address with different priorities. If memory management is not enabled, all transactions will be considered low priority.

DRAM Controller

There are two DRAM queues, which allows transactions to be split into high and low priority with high-priority transactions being handled before low-priority. Transactions are placed in the queue according to their GTLB priority setting.

The number of DRAM banks is determined by the memory configuration set in Cerberus octlet 10. The DRAM controller accepts two queue requests for each DRAM bank.

Hermes Controllers

All transactions to Hermes will be treated with a single priority and handled in the order they arrive from their respective queues. The Hermes controllers accept up to eight queue requests each.

Slow Peripheral Controller

Query: *description of slow per. con.*

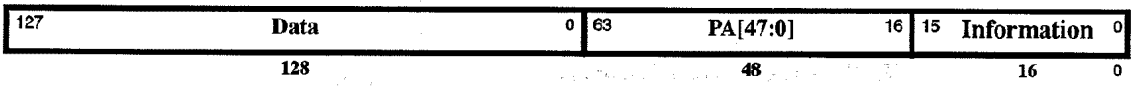
All transactions the slow devices (such as GTLB) will be treated with a single priority and handled in the order they arrive from their respective queues. The Slow Peripheral controller handles only one queue request at a time.

Query: *description of ready signal*

MU 0054537

Format of a Non-blocking Entry

A non-blocking entry has three parts, the data, physical address, and additional information which is described in the table below.



Bits	Information field
15..14	Reserved
13	No allocate
12..10	Cylinder number
9..7	Size code [a]
6	Set to indicate big-endian, otherwise it is little-endian

5	Set to indicate the data is signed
4	Set to indicate icache, otherwise it is dcache
3	Set to indicate high priority, otherwise it is low priority
2	Set to indicate a cache request
1	Set to indicate store
0	Set to indicate load

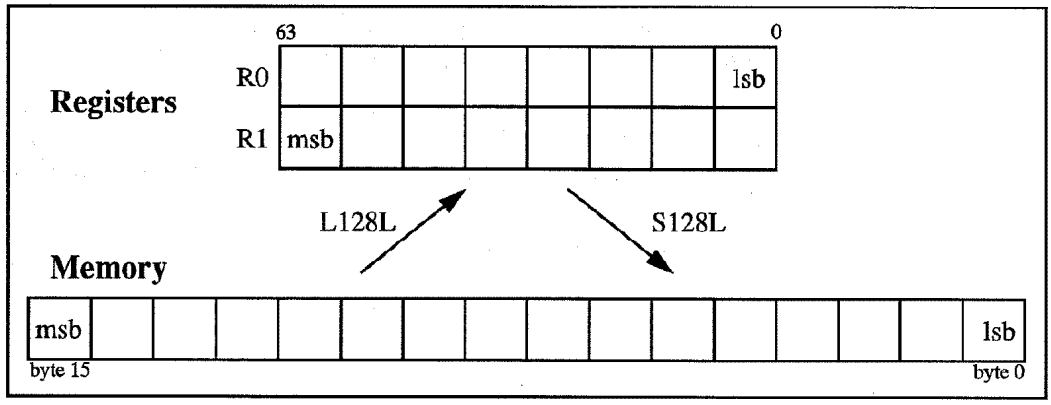
a. Three bits make up the memory size code: 001 = byte; 010 = doublet; 011 = quadlet; 000 = octlet; 100 = hexlet. 101, 110, and 111 do not occur.

The non-blocking load buffer entry is not directly software accessible. Additional important information, the register number or cache index to which the data returns, is a part of a NB transaction but is stored elsewhere.

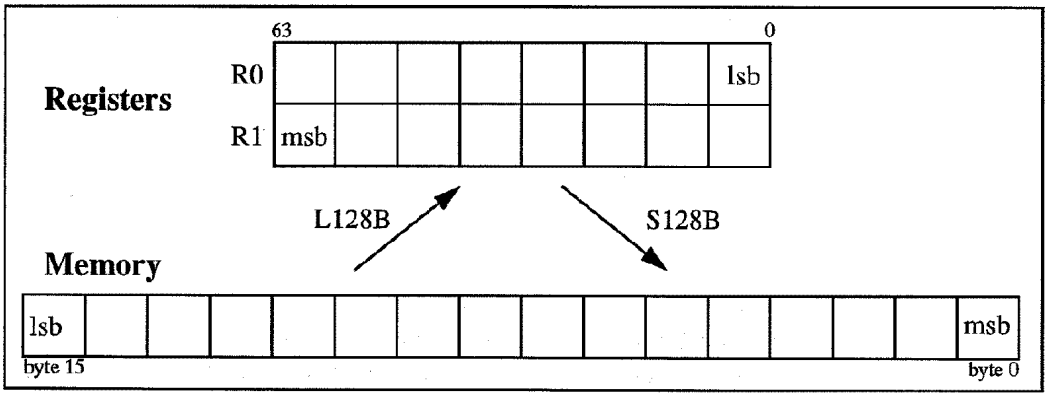
Writing/Reading Memory (Endianess)

Data can be loaded and stored either *little-endian* or *big-endian*. Each can best be described by a picture showing how the data is mapped using, for example, a store and load instruction.

Little-endian (128-bit)

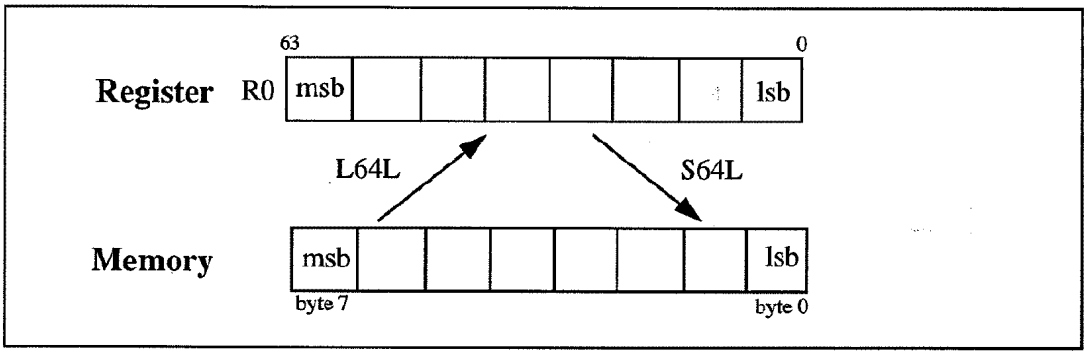


Big-endian (128-bit)

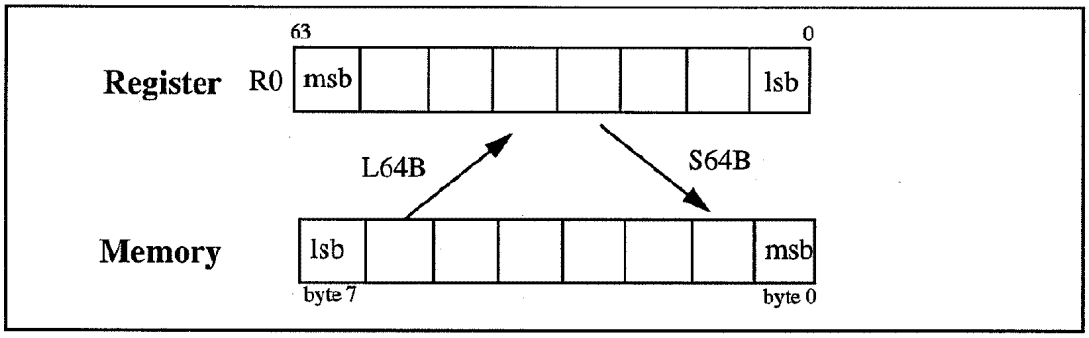


MU 0054539

Little-endian (64-bit)



Big endian (64-bit)



MU 0054540

Highly Confidential

Chapter 5

Event Handling

MU 0054541

There are two types of events recognized by Euterpe: exceptions and interrupts.

Exceptions signal several kinds of events: (1) events that indicate failure of the software or hardware, such as arithmetic overflow, (2) events that are hidden from the virtual process model, such as translation buffer misses, (3) events that occur infrequently, but may require corrective action. See the list of exceptions on page 46.

Interrupts are a means of communication and cylinder control. There are three type of interrupts (1) I/O interrupts, (2) Timer interrupts, and (3) inter-cylinder signalling interrupts. Interrupts are described in more detail below.

Each of these types of events require the interruption of the current flow of execution, handling of the exception or interrupt, and in some cases descheduling of the current task and rescheduling of another.

When a cylinder handles an event, all of the cylinder's resources are available for the handling of the event, except for the ability of that cylinder to handle additional events. Euterpe prevents additional interrupts, to the affected cylinder, from being recognized until the cylinder returns from its interrupt handler. Only one exception, for all of Euterpe, is handled at a time. Other cylinders continue their normal execution, and can

also handle interrupts. If another cylinder encounter an exception, however, its execution is stalled until the first cylinder returns from handling its exception. Additional exceptions to the affected cylinder, prior to its return, cause a machine check. See page 53 for more information about machine checks.

Event Handling Mechanism

The event mechanism is similar for exceptions and interrupts. The first 160 bytes of the data buffer are reserved for each cylinder's new and old program counter (PC) and register 1 (R1) contents.

offset	contents
0x00	Pre-Event PC and Privilege for Cylinder 0
0x08	Pre-Event R1 for Cylinder 0
0x10	Post-Event PC and Privilege for Cylinder 0
0x18	Post-Event R1 for Cylinder 0
0x20	Pre-Event PC and Privilege for Cylinder 1
0x28	Pre-Event R1 for Cylinder 1
0x30	Post-Event PC and Privilege for Cylinder 1
0x38	Post-Event R1 for Cylinder 1
0x40	Pre-Event PC and Privilege for Cylinder 2
0x48	Pre-Event R1 for Cylinder 2
0x50	Post-Event PC and Privilege for Cylinder 2
0x58	Post-Event R1 for Cylinder 2
0x60	Pre-Event PC and Privilege for Cylinder 3
0x68	Pre-Event R1 for Cylinder 3
0x70	Post-Event PC and Privilege for Cylinder 3
0x78	Post-Event R1 for Cylinder 3
0x80	Pre-Event PC and Privilege for Cylinder 4
0x88	Pre-Event R1 for Cylinder 4
0x90	Post -Event PC and Privilege for Cylinder 4
0x98	Post-Event R1 for Cylinder 4

The privilege level occupies the least significant 2 bits of the *Pre-Event PC*

and Privilege fields.

When an exception occurs, the pc of the current instruction, with the current privilege level, and the contents of register 1 are saved in the appropriate cylinder's *Pre-Event PC and Privilege* and R1 fields. When an interrupt is recognized, the same is true, except the pc is set for the next instruction.

The cylinder enters Event Mode, which prevents the cylinder from receiving another event until it returns from handling the current event. The cyllinder's Post-Event pc, privilege, and R1 contents are loaded, and execution begins in Event Mode.

The BBACK instruction is used to return from event handling. This instruction causes the Pre-Event pc and privilege, and the Pre-Event R1 contents to be loaded, and the cylinder exits event mode. The event handler, subject to constraints imposed by memory management, may alter the Pre-Event pc, privilege, and R1 values.

System Registers in Events

Several system registers are involved in causing and handling events.

If the event is an exception, the Exception status register is loaded. If the exception is an addressing exception, the Exception address register is also loaded.

If the event is an interrupt, a bit is set in the Event Register. Which event bits are recognized, on a per-cylinder basis, is controlled by the Event Mask Register. Other registers get involved in setting the bits in the Event Register, namely the Event Daemon Access Register, and the Timer and Timer Mask Registers. The Event Register is also writable.

The functions of each of these registers are described in more detail on page 37.

Chapter 6

Reset and Error Recovery

Some internal and external events cause the Euterpe processor to invoke a reset, logic clear, or machine check. One example of these events is the initial power up of the processor, which automatically causes a reset.

These operations consist of a full or partial reset of critical machine state, including initialization of each cylinder to begin fetching instructions from the Start Vector Address (see page 57).

Software may determine whether a reset, logic clear, or machine check has occurred by reading the value of the Cerberus control register (register 6):

- if the *reset* and *clear* bits are set, a reset has occurred
- if the *reset* and *clear* bits are cleared, a machine check has occurred
- if the *reset* bit is cleared, and the *clear* bit is set, a logic clear has occurred.

When either a reset or machine check is indicated, the contents of the Cerberus status register (register 7) contain more detailed information on the cause.

MU 0054544

Highly Confidential

Reset

A reset is caused by Euterpe power up, the detection of an over-temperature condition, or a hardware error. A reset sets the Cerberus registers to their default states, performs a circuit reset and then performs a logic clear.

Power Up

At power up a reset is automatically performed. The default knob settings in the Cerberus registers are nominal so that Euterpe will warm up faster.

Under certain extreme temperature and low voltage conditions the beginning of reset will be delayed until the chip has warmed up enough to be operational.

Over-temperature Condition

An over-temperature condition is a special example of a hardware-detected error. An over-temperature reset overrides the default geographical digital knob settings in the Cerberus registers, setting them to the lowest power levels. The knob settings are found in Cerberus registers 21 through 29.

Hardware Error

If the reset is caused by a hardware error, there will be additional information about the cause of the reset in Cerberus register 7. This includes:

Over-temperature (Bit 61)

If Euterpe detects an over-temperature condition, it resets itself to a minimum power state.

Double Machine Check (Bit 60)

MU 0054545

If two machine checks occur, and the second machine check occurs before the first machine check is acknowledged (by writing a value of 0 to Cerberus register 7), Euterpe will reset.

Logic Clear

A logic clear is performed by the hardware during a reset. It can also be triggered by setting the *clear* bit in Cerberus register 6.

Effect

Logic clear resets the ECL logic, but does not change the Cerberus registers, except as noted below. However, the state of register file, system registers, tlbs, tags, and buffers is undefined. The state of DRAM can be preserved if the interface has been fenced off prior to the the logic clear. To fence off the interface, set the *output set high* bit in Cerberus register 10.

Bits 55:0 in Cerberus register 6 are set to their default state, disabling memory management and Hermes channels. The logic clear also transfers any deferred writes into the Cerberus registers.

After a logic clear, each cylinder comes up in *event mode*. There can be no events in a cylinder until software in that cylinder executes a BBACK instruction. The event register, event mask register, the timer register, and the timer match register must be initialized before the BBACK.

The watchdog machine check will be disabled by a logic clear until the first write to the watchdog register.

Duration

The logic clear causes a concurrent PLL and logic reset. The PLL reset lasts 4607 Cerberus cycles while the logic reset lasts 9215 Cerberus cycles.

About 50 Cerberus clocks after a logic clear completes, the Hermes channels can be enabled. This time is necessary for the interface to synchronize clocks with any devices on the channel.

Machine Check

MU 0054546

Detected hardware errors invoke a machine check.

While it is possible that much of the state of the machine, for example the main register files, would be preserved after a machine check, this is not

guaranteed. Therefore, it is safest to consider the state after a machine check to be similar to the state after a logic clear.

Hardware Error

The cause of a machine check is indicated in Cerberus register 7. A write of zeros to register 7 acknowledges the machine check and clears the machine check cause bits. Causes include:

Exception in event mode (Bit 58)

This bit indicates that a cylinder encountered an exception while in *event mode*.

Watchdog time-out error (Bit 57)

This bit indicates that a machine check occurred because the contents of the watchdog match register equaled the cycle count. This machine check can occur only after the watchdog match register has been initialized.

Cerberus transaction error (Bit 56)

This bit indicates that Euterpe detected an error on the Cerberus bus. This includes parity errors and time-outs.

Hermes channel check byte error (Bit 55)

This bit indicates that Euterpe encountered a check byte error in a packet on an enabled Hermes channel. The *machine check detail* field in Cerberus register 7 indicates which Hermes channel encountered the error.

Hermes channel command error (Bit 54)

This bit indicates that Euterpe encountered an illegal command in a packet on an enabled Hermes channel. The *machine check detail* field in Cerberus register 7 indicates which Hermes channel encountered the error.

Hermes channel time-out error (Bit 53)

This bit indicates that Euterpe encountered a watchdog time-out when Hermes transactions were pending.

Start Vector Address

When a power up, reset, logic clear, or machine check occurs, each Euterpe cylinder begins executing at the Start Vector Address. There is only one Start Vector Address for all the cylinders.

The Start Vector Address normally points to the first instruction in the Programmable ROM. If Euterpe's SN3 pin is pulled high, the Start Vector Address is in Cerberus Space.

Power Down

Euterpe's power consumption and clock speed are programmable through Cerberus registers. In *power down* mode, Euterpe can be programmed to operate at minimum power by setting the appropriate Cerberus registers. However, even at the lowest power with default clock settings, Euterpe is still functional.

Chapter 7

Cerberus Registers

MicroUnity's Cerberus serial bus architecture is designed to provide bootstrap resources, configuration and diagnostic support for MicroUnity's Terpsichore system architecture.

Cerberus registers are internal read-only and read-write registers which provide a mechanism to query and control the configuration of the devices in the system. By use of these registers, a user of the system may tailor the use of the facilities for maximum performance and utility. Conversely, a supplier of a system component may modify facilities in the device without compromising compatibility with earlier implementations. These registers are accessed via the Cerberus serial bus.

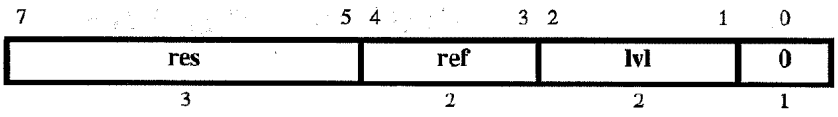
Euterpe implements Cerberus as it is described in *MicroUnity Terpsichore System Architecure*. See the section Cerberus Serial Bus in that document for a more detailed explanation of how Cerberus works.

MU 0054549

Power and Swing Calibration Registers

Euterpe uses a set of calibration registers to control the power and voltage levels used for internal high-bandwidth logic and memory. Eight-bit fields separately control the power and voltage levels used in a portion of the

Euterpe circuitry. Each such field used to control digital circuitry (labeled *knob*) contains configuration data in the following format:



Please see the section Power and Swing Calibration Registers in *Terpsichore System Architecture* for information on the fields in these registers.

Cerberus Registers

The Euterpe-implemented format of the Cerberus registers is described in the table below. If a reset does not initialize the field to a value, or if initialization is not required by this specification, a * is placed in or appended to the value field.

octlet 0	bits	field name	value (hex)	range	interpretation
	63 : 16	**architecture code**	0x00 40 a3 24 69 93		Identifies processor device as compliant with MicroUnity Euterpe processor architecture.
	15 : 0	**architecture revision**	01 00		Device complies with architecture version 1.0.

octlet 1	bits	field name	value (hex)	range	interpretation
	63 : 16	**implementor code**	0x00 40 a3 d2 b6 7f		Identifies Euterpe processor device as implemented by MicroUnity. MU 0054551
	15 : 0	**implementor revision**	01 00		Implementation version 1.0.

octlet 2	bits	field name	value (hex)	range	interpretation
	63 : 16	**manufacturer code**	0x00 40 a3 69 db 3f		Identifies initial manufacturer of Euterpe processor device implemented by MicroUnity as MicroUnity.
	15 : 0	**manufacturer revision**	01 00		Manufacturing version 1.0.

octlet 3	bits	field name	value	range	interpretation
	63 : 16	**serial number**	0		This device has no serial number capability.
		dynamic address	0		This device has no dynamic addressing capability.

octlet 4	bits	field name	value	range	interpretation
	63 : 60	**A**	4	0..15	Size of a Hermes address
	59 : 56	**$\log_2 W$**	3	0..15	Size of a Hermes word
	55 : 0	**0**	0	0	Reserved for definition in later revision of Euterpe architecture

octlet 5	bits	field name	value	range	interpretation
	63 : 0	**0**	0	0	Reserved for definition in later revision of Euterpe architecture

octlet 6	bits	field name	value	range	interpretation
	63	**reset**	1	0..1	Set to invoke device's circuit reset
	62	**clear**	1	0..1	Set to invoke device's logic clear
	61	**selftest**	0	0..1	Set to invoke device's selftest: bits 60..48 may indicate depth of selftest
	60	**defer writes**	0*	0..1	Set to cause writes to octlets 21..33 to be deferred until the next logic-clear or non-deferred write
	59 : 53	**0**	0	0	Reserved
	52 : 48	**NB priority**	16	0..16	Set to the number of non-blocking entries that can be used by low-priority requests
	47 : 34	**0**	0	0	Reserved for additional Hermes channel disable bits
	33	**Hermes channel 1 disable**	1	0..1	Set to cause input channel 1 to be ignored and idles to be generated. Upon clearing the bit, the input channel phase adjustment is reset, and after a suitable delay, the input and Hermes output channel links are available for use.
	32	**Hermes channel 0 disable**	1	0..1	Set to cause input channel 0 to be ignored and idles to be generated. Upon clearing the bit, the input channel phase adjustment is reset, and after a suitable delay, the input and Hermes output channel links are available for use.
	31	**0**	0	0	Reserved
	30	**memory management enable**	0	0	Enable memory management

octlet 6	bits	field name	value	range	interpretation
	29 : 25	**icache [a] configuration**	0	*n*	Set instruction cache size to 2^*n* bytes. The following are legal 5-bit values for *n* and their interpretation: 0: no cache (32K buffer) 12: 4K cache (28K buffer) 13: 8K cache (24K buffer) 14: 16K cache (16K buffer)
	24 : 20	**dcache [a] configuration**	0	*n*	Set data cache size to 2^*n* bytes. The following are legal 5-bit values for *n* and their interpretation: 0: no cache (32K buffer) 12: 4K cache (28K buffer) 13: 8K cache (24K buffer) 14: 16K cache (16K buffer)
	19 : 16	**channel under test**	0*	0..1	Channel on which cidle 0 and cidle 1 are transmitted in place of normal idle pattern (0, 255), and from which raw input bytes are sampled
	15 : 8	**cidle 0**	0*	0..255	Value transmitted on idle Hermes output channel when output clock zero (0).
	7 : 0	**cidle 1**	255*	0..255	Value transmitted on idle Hermes output channel when output clock one (1).

a. Changes to the cache configuration take effect immediately.

MU 0054553

octlet 7[a]	bits	field name	value	range	interpretation
	63	**reset/clear/ selftest complete**	1	0..1	This bit is set when a reset, clear or selftest operation has been completed.
	62	**reset/clear/ selftest status**	1	0..1	This bit is set when a reset, clear or selftest operation has been completed successfully.
	61	**meltdown detected**	0	0..1	This bit is set when the meltdown detector has caused a reset.
	60	**double machine check**	0	0..1	This bit is set when a double machine check has caused a reset.
	59	**other reset cause**	0	0..1	This bit is reserved for indicating additional causes of reset.
	58	**exception in exception mode**	0	0..1	This bit is set when an exception occurs and the thread was already in exception mode.
	57	**watchdog timeout error**	0	0..1	This bit is set when a watchdog timeout has caused a machine check.
	56	**Cerberus transaction error**	0	0..1	This bit is set when a Cerberus transaction error or parity error has caused a machine check.
	55	**Hermes channel check byte error**	0	0..1	This bit is set when a Hermes channel check byte error has caused a machine check.
	54	**Hermes channel command error**	0	0..1	This bit is set when a Hermes channel command error has caused a machine check.
	53	**Hermes channel timeout error**	0	0..1	This bit is set when a watchdog timeout error has occurred (bit 57) and there is a pending Hermes transaction.
	52 : 48	**0**	0*	0	Reserved for other machine check causes.
	47 : 32	**machine check detail**	0*	0..3	Indicates which Hermes channel encountered an error.
	31 : 16	**0**	0	0	Reserved
	15 : 8	**raw 0**	*	0..255	Value sampled on specified Hermes channel when input clock is zero (0).
	7 : 0	**raw 1**	*	0..255	Value sampled on specified Hermes channel immediately following sample value in **raw 0** register.

a. To clear the bits in octlet 7 after a machine check, you use an octlet store of all 0x0. Storing any other values is undefined.

MU 0054554

Highly Confidential

octlet 8..9	bits	field name	value	range	interpretation
	63 : 0	**0**	0	0	Reserved for additional access to physical addresses

octlet 10[a]	bits	field name	value	range	interpretation
	63 : 56	**0**	0	0	Reserved
	55	**preasap**	1	0..1	Precharge as soon as possible
	54	**bsa 20**	0	0..1	Bank select from address bit 20
	53	**restart**	0	0..1	Rising edge initiates DRAM part reset sequence and mode register load.
	52	**refresh enable**	0	0..1	Enables automatic refreshing
	51	**sleep**	0	0..1	Rising edge initiates transition to clock suspend mode; falling edge initiates transition from clock suspend to normal operating mode. Should be 0 at restart time. Note: Be careful to write restart bit to 0 so it won't reset part and reload Mode register; and be careful not to change other control bits.
	50 : 46	**ratio m1**	12	3..31	(*ratio m1* + 1) is the ratio of SOFA clock to DRAM clock frequency. Counting nominal SOFA clock frequency as 1296 MHz, nominal value the ratio is 13, with a range from 4 to 32, sdclock frequency is ~99.7 MHz.
	45 : 38	**refresh mult m1**	97	0..255	Do an autorefresh sequence every (*ratio m1* + 1)*16*(*refresh mult m1* + 1) SOFA clock cycles. There's a built in prescalar of 16.
	37	**output set high**	0	0..1	Sets all DR interface signals to logical high.[b]
	36	**clock invert**	0	0..1	Invert sdclock waveform (negative edge is first after control/address/data changes)
	35 : 31	**clock offset p1**	6	1..*ratio m1*	(*clock offset p1* - 1) is the number of SOFA clock periods between the control/address/data changing and the next edge of sdclock; rising for *clock invert* = 0; falling for *clock invert* = 1.
	30 : 26	**clock half-period**	6	1..value	Time clock is high (*clock invert* = 0) or low (*clock invert* = 1). value is: (*ratio m1* + 1 - *clock offset p1*)
	25 : 21	**sample offset p1**	5	0..*ratio m1*	(*sample offset p1* - 1) is the number of SOFA clock periods between the control/address/data changing and when incoming data will be sampled. A value of 0 means sampling will occur 1 clock before control/address/data.[c]
	20 : 19	**tRCD**	3	1..max[d]	Time of the tRCD parameter in DRAM clock cycles coded as integers.
	18 : 17	**tWR**	2	1..max[d]	Time of the tWR parameter in DRAM clock cycles coded as integers.

octlet 10[a]	bits	field name	value	range	interpretation
	16 : 14	**tAA**	3	1..max [d]	Time of the tAA parameter in DRAM clock cycles coded as integers. tAA (CAS latency) goes into the Mode register as is.
	13 : 11	**tRP**	4	1..max [d]	Time of the tRP parameter in DRAM clock cycles coded as integers.
	10 : 8	**tRRD**	3	1..max [d]	Time of the tRRD parameter in DRAM clock cycles coded as integers.
	7 : 4	**tRCm1**	9	1..max [d]	Time of the tRCm1 parameter in DRAM clock cycles coded as integers. tRCm1 (tRC-1) generally is not unique, but is input to save computation.
	3	**part 64m**	0	0..1	Value indicates parts are 1 = 64Mbit or 0 = 16 Mbit.
	2	**bl4**	0	0..1	Parts configuration uses: 0 = all 32 data pins, consequently burst length is 2; 1 = only 16 of 32 data pins, so burst length is 4. Burst Length (coded) goes into the Mode register.
	1 : 0	**part width**	0	0..1	Part width code: 0b00 means by-16; 0b01 means by-8; 0b10 means by-4; 0b11 is illegal and not used.

a. All registers are read-write. The *restart* bit (53) only has an effect when it is written from 0 to 1. The *sleep* bit (51) only has an effect when it changes. Other bits are effective immediately, but you probably don't want to change much without restarting the sdram. So a normal sequence would be to write configuration, electrical and logical parameters with restart 0, then write the same parameters with restart 1 to restart the sdram and reload the Mode register. For more information on Euterpe DRAM, see page 34.

b. Sets all DR interface signals to logical high, even if sdddrive would indicate bidirectional data i/o's are not being driven. Since this signal originates in CMOS SOFA, it is immune to logic level disruptions occurring in ECL SOFA during power level changes. A power-down sequence would first set *sleep* high, then set *output set high* to high, then power levels can be changed without disrupting DRAM data or putting spurious signals on the DRAM pins. The power up sequence would first ensure that power levels are stable, then set *output set high* to low, then set *sleep* to low. If no timing parameters have changed, the DRAM will be ready for read and write. If timing parameters have changed, these will have to be written to appropriate registers and the *restart* bit changed from low to high to reload the Mode register.

c. Note that a zero or unit delay logical simulation in which the data sample point follows the sdclock rising edge will fail. Although in the high-speed nominal configuration the data sample point will follow sdclock rising edge because the delay through the part and buffers is greater than one sdclock period.

d. Legal values are 1:max but should correspond to timing parameter's actual values and actual clock period. When the part is operating from a very low frequency clock, the values will be 1, except for tRCm1=0. tCCD is implied and is always 2.

octlet 11..20	bits	field name	value	range	interpretation
	63 : 0	**0**	0	0	Reserved for expansion of Cerberus registers upward or knobcity registers downward.

octlet 21 [a]	bits	field name	value	range	interpretation
	63 : 56		224	0..255	iobyte0 quadrature
	55 : 48		224	0..255	Geographical digital knob settings
	47 : 40		224	0..255	Geographical digital knob settings
	39 : 32		224	0..255	Geographical digital knob settings
	31 : 24		224	0..255	Geographical digital knob settings
	23 : 16		224	0..255	iobyte0 drive current
	15 : 8		0	0..255	unused
	7 : 0		0	0..255	unused

a. If a reset occurs due to an over-temperature condition, Euterpe overrides the reset defaults of the knob settings to a value of 1 (minimum power).

octlet 22 [a]	bits	field name	value	range	interpretation
	63 : 56		224	0..255	north ttl buffers
	55 : 48		224	0..255	Geographical digital knob settings
	47 : 40		224	0..255	Geographical digital knob settings
	39 : 32		224	0..255	Geographical digital knob settings
	31 : 24		224	0..255	Geographical digital knob settings
	23 : 16		224	0..255	iobyte0 generic bias
	15 : 8		0	0..255	unused
	7 : 0		0	0..255	unused

a. If a reset occurs due to an over-temperature condition, Euterpe overrides the reset defaults of the knob settings to a value of 1 (minimum power).

octlet 23 [a]	bits	field name	value	range	interpretation
	63 : 56		224	0..255	dcache, tag
	55 : 48		224	0..255	Geographical digital knob settings
	47 : 40		224	0..255	Geographical digital knob settings
	39 : 32		224	0..255	Geographical digital knob settings
	31 : 24		224	0..255	Geographical digital knob settings
	23 : 16		224	0..255	Reserved knob
	15 : 8		0	0..255	unused
	7 : 0		0	0..255	unused

a. If a reset occurs due to an over-temperature condition, Euterpe overrides the reset defaults of the knob settings to a value of 1 (minimum power).

MU 0054557

Highly Confidential

octlet 24 [a]	bits	field name	value	range	interpretation
	63 : 56		224	0..255	pll1
	55 : 48		224	0..255	Geographical digital knob settings
	47 : 40		224	0..255	Geographical digital knob settings
	39 : 32		224	0..255	Geographical digital knob settings
	31 : 24		224	0..255	Geographical digital knob settings
	23 : 16		224	0..255	Geographical digital knob settings
	15 : 8		224	0..255	Geographical digital knob settings
	7 : 0		224	0..255	south ttl buffers

a. If a reset occurs due to an over-temperature condition, Euterpe overrides the reset defaults of the knob settings to a value of 1 (minimum power).

octlet 25 [a]	bits	field name	value	range	interpretation
	63 : 56		224	0..255	gtlb
	55 : 48		224	0..255	Geographical digital knob settings
	47 : 40		224	0..255	Geographical digital knob settings
	39 : 32		224	0..255	Geographical digital knob settings
	31 : 24		224	0..255	Geographical digital knob settings
	23 : 16		224	0..255	Reserved knob
	15 : 8		0	0..255	unused
	7 : 0		0	0..255	unused

MU 0054558

a. If a reset occurs due to an over-temperature condition, Euterpe overrides the reset defaults of the knob settings to a value of 1 (minimum power).

octlet 26 [a]	bits	field name	value	range	interpretation
	63 : 56		224	0..255	pll0, special clock input buffer
	55 : 48		224	0..255	Geographical digital knob settings
	47 : 40		224	0..255	Geographical digital knob settings
	39 : 32		224	0..255	Geographical digital knob settings
	31 : 24		224	0..255	Geographical digital knob settings
	23 : 16		224	0..255	Geographical digital knob settings
	15 : 8		224	0..255	Geographical digital knob settings
	7 : 0		224	0..255	CMOS register file

a. If a reset occurs due to an over-temperature condition, Euterpe overrides the reset defaults of the knob settings to a value of 1 (minimum power).

octlet 27 [a]	bits	field name	value	range	interpretation
	63 : 56		224	0..255	icache, tag
	55 : 48		224	0..255	Geographical digital knob settings
	47 : 40		224	0..255	Geographical digital knob settings
	39 : 32		224	0..255	Geographical digital knob settings
	31 : 24		224	0..255	Geographical digital knob settings
	23 : 16		224	0..255	clock tree
	15 : 8		0	0..255	unused
	7:0		0	0..255	unused

a. If a reset occurs due to an over-temperature condition, Euterpe overrides the reset defaults of the knob settings to a value of 1 (minimum power).

octlet 28 [a]	bits	field name	value	range	interpretation
	63 : 56		224	0..255	stack sensor iobyte1 generic bias
	55 : 48		224	0..255	Geographical digital knob settings
	47 : 40		224	0..255	Geographical digital knob settings
	39 : 32		224	0..255	Geographical digital knob settings
	31 : 24		224	0..255	Geographical digital knob settings
	23 : 16		224	0..255	iobyte1 quadrature
	15 : 8		0	0..255	unused
	7 : 0		0	0..255	unused

a. If a reset occurs due to an over-temperature condition, Euterpe overrides the reset defaults of the knob settings to a value of 1 (minimum power).

octlet 29 [a]	bits	field name	value	range	interpretation
	63 : 56		224	0..255	Reserved
	55 : 48		224	0..255	Geographical digital knob settings
	47 : 40		224	0..255	Geographical digital knob settings
	39 : 32		224	0..255	Geographical digital knob settings
	31 : 24		224	0..255	Geographical digital knob settings
	23 : 16		224	0..255	iobyte1 drive current
	15 : 8		0	0..255	unused
	7 : 0		0	0..255	unused

MU 0054559

a. If a reset occurs due to an over-temperature condition, Euterpe overrides the reset defaults of the knob settings to a value of 1 (minimum power).

octlet 30 bits	field name	value	range	interpretation
63 : 62	**Hermes skew lvl**	0	0..3	Swing control field (lvl) for Hermes skew control circuits
61 : 60	**0**	0	0	Reserved
59 : 57	**global resistor**	7	0..7	Global resistor (power) setting
56	**resistor select**	0	0..1	Mux selection (enable). Default value of 0 indicates that the resistor setting will be taken from the pads. If the value is changed to 1, the value will be taken from the global resistor setting.
55 : 53	**0**	0	0	Reserved
52 : 48	**termination fine-tuning**	20	0..31	Set based on value from PMOS drive strength, used to fine-tune resistor values in Hermes termination
47 : 45	**0**	0	0	Reserved
44 : 40	**process control**	20	16..31	Set based on value read from PMOS drive strength, used to fine-tune resistor values in knob settings
39 : 37	**0**	0	0	Reserved
36 : 32	**PMOS drive strength**	20*	12..27	This read-only field indicates the drive strength of PMOS devices expressed as a digital binary value.
31 : 28	**swing 3**	15	0..15	Voltage swing knob setting 3
27 : 24	**swing 2**	15	0..15	Voltage swing knob setting 2
23 : 20	**swing 1**	15	0..15	Voltage swing knob setting 1
19 : 16	**swing 0**	15	0..15	Voltage swing knob setting 0
15 : 12	**reference 3**	15	0..15	Voltage reference knob setting 3
11 : 8	**reference 2**	15	0..15	Voltage reference knob setting 2
7 : 4	**reference 1**	15	0..15	Voltage reference knob setting 1
3 : 0	**reference 0**	15	0..15	Voltage reference knob setting 0

octlet 31 bits	field name	value	range	interpretation
63 : 48	**0**	0	0	Reserved
47 : 43	**PLL1 divide ratio**	5*	3..13	PLL1 divider ratio. The value is doubled when sent through Hermes (range 6..26).
42	**PLL1 feedback bypass**	1*	0..1	Set to invoke PLL1 feedback bypass
41	**PLL1 range**	0*	0..1	Set for operation at high frequency (above 0.45 GHz); cleared for operation at low frequency (below 0.45 GHz). Used for ring oscillator only.
40	**PLL1 oscillator select**	0	0..1	Set to select multivibrator oscillator. Cleared to select ring oscillator
39 : 35	**PLL0 divide ratio**	10	3..13	PLL0 divider ratio. The value is doubled when sent to the SOFA clock (range 6..26)
34	**PLL0 feedback bypass**	1	0..1	Set to invoke PLL0 feedback bypass
33	**PLL0 range**	0	0	Set for operation at high frequency (above 0.45 GHz); cleared for operation at low frequency (below 0.45 GHz). Used for ring oscillator only.
32	**PLL0 oscillator select**	0	0	Set to select multivibrator oscillator. Cleared to select ring oscillator
31 : 24	**analog measurement**	0	0	Set to measure analog levels at various test points within device
23 : 22	**meltdown threshold**	0	0..3	Set to perform margin testing of the meltdown detector. Each value corresponds to the following testing temperature: 0 - 155 degrees C 1 - 60 degrees C 2 - 100 degrees C 3 - 25 degrees C
21	**conversion start**	0*	0..1	Setting this bit causes the conversion to begin. The bit remains set until conversion is complete
20 : 17	**0**	0	0	Reserved (selection extension)
16	**conversion selection**	0*	0..1	This field selects which conversion to perform, either temperature or reference. The results can be used to compute absolute temperature.
15 : 10	**0**	0	0	Reserved (counter extension)
9 : 0	**conversion counter**	0*	400..700	This field is set to the two's complement of the downslope count. The counter counts upward to zero, and then continues counting on the upslope until conversion completes. The temperature range is from 0 to 125 degrees C.

MU 0054561

octlet 32[a] bits	field name	value	range	interpretation
63	**0**	0	0	Reserved
62	**quadrature bypass**	0*		Setting this bit causes the quadrature circuit to be bypassed; the input clock signal is used directly.
61	**quadrature range**	0*		Set to 0 if the Hermes channel is operating at a low frequency; 1 if the Hermes channel is operating at a high frequency
60	**output termination**	1		Set to enable output terminators. Clear to disable output terminators.
59 : 57	**termination resistance**	1		Set termination resistance level
56 : 54	**output current**	1		Set output current level **MU 0054562**
53 : 48	**skew bit 7**	1		Set delay in Ho7 skew circuit
47 : 42	**skew bit 6**	1		Set delay in Ho6 skew circuit
41 : 36	**skew bit 5**	1		Set delay in Ho5 skew circuit
35 : 30	**skew bit 4**	1		Set delay in Ho4 skew circuit
29 : 24	**skew bit 3**	1		Set delay in Ho3 skew circuit
23 : 18	**skew bit 2**	1		Set delay in Ho2 skew circuit
17 : 12	**skew bit 1**	1		Set delay in Ho1 skew circuit
11 : 6	**skew bit 0**	1		Set delay in Ho0 skew circuit
5 : 0	**skew clk**	1		Set delay in HoC skew circuit

a. Register 32 controls Hermes channel 0.

octlet 33[a] bits	field name	value	range	interpretation
63	**0**	0	0	Reserved
62	**quadrature bypass**	0*		Setting this bit causes the quadrature circuit to be bypassed; the input clock signal is used directly.
61	**quadrature range**	0*		Set to 0 if the Hermes channel is operating at a low frequency; 1 if the Hermes channel is operating at a high frequency
60	**output termination**	1		Set to enable output terminators. Clear to disable output terminators.
59 : 57	**termination resistance**	1		Set termination resistance level
56 : 54	**output current**	1		Set output current level
53 : 48	**skew bit 7**	1		Set delay in Ho7 skew circuit
47 : 42	**skew bit 6**	1		Set delay in Ho6 skew circuit
41 : 36	**skew bit 5**	1		Set delay in Ho5 skew circuit

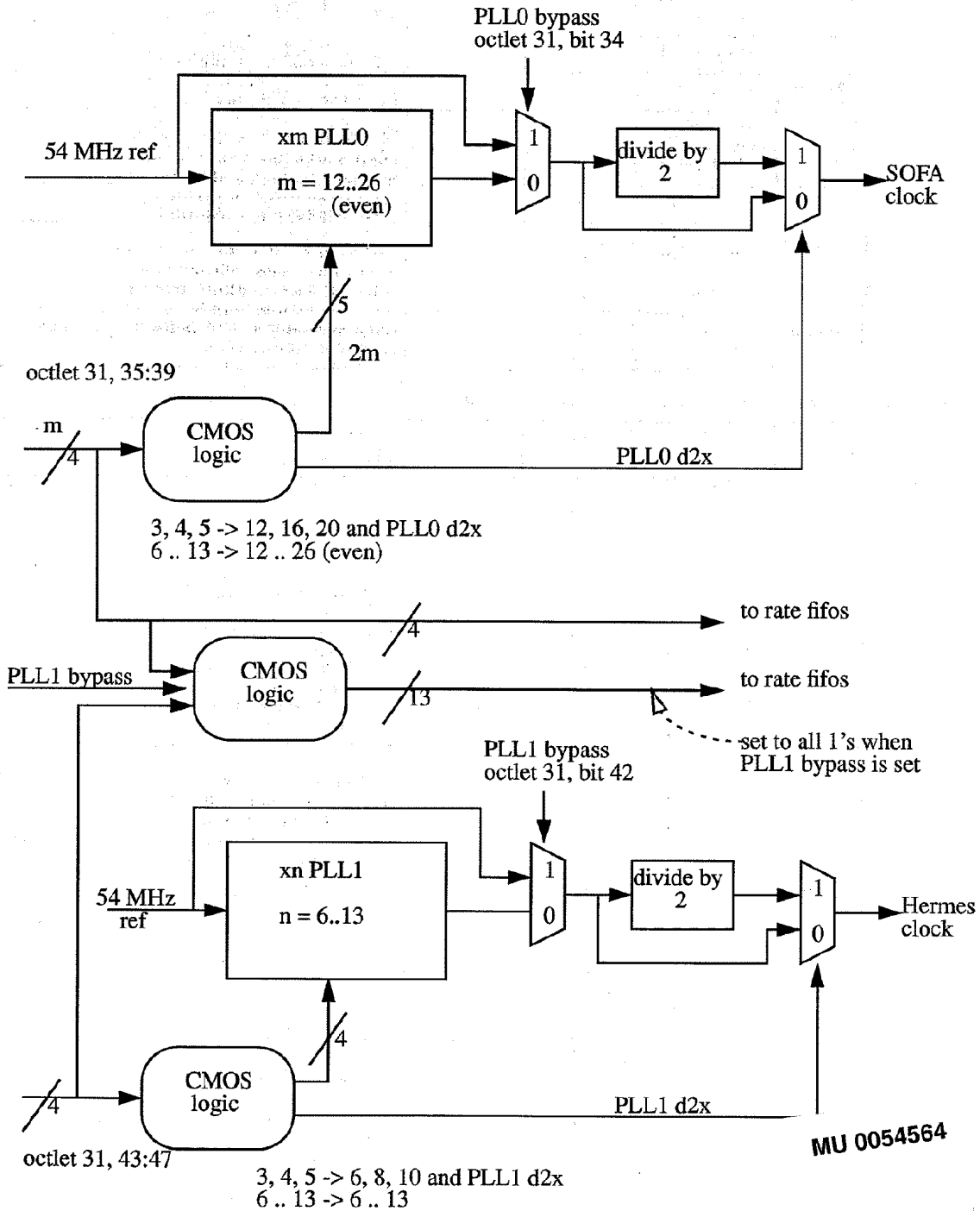
octlet 33[a]	bits	field name	value	range	interpretation
	35 : 30	**skew bit 4**	1		Set delay in Ho4 skew circuit
	29 : 24	**skew bit 3**	1		Set delay in Ho3 skew circuit
	23 : 18	**skew bit 2**	1		Set delay in Ho2 skew circuit
	17 : 12	**skew bit 1**	1		Set delay in Ho1 skew circuit
	11 : 6	**skew bit 0**	1		Set delay in Ho0 skew circuit
	5 : 0	**skew clk**	1		Set delay in HoC skew circuit

a. Register 33 controls Hermes channel 1.

octlet 34..63	bits	field name	value	range	interpretation
	63 : 0	**0**	0	0	Reserved for use with additional Hermes channel interfaces

octlet 64...	bits	field name	value	range	interpretation
	63 : 0	**0**	0	0	Reserved for user with later revisions of the architecture

Clock Generation Scheme



MU 0054564

Highly Confidential

Temporary Change file

This file will be turned into a change file for each of the more official versions of the book (the ones with new release numbers).

List of changes: 1.3 - 1.4

Some more formatting changes throughout the book.

- opcodes.mif
  - — EGFMUL64 is not being implemented
  - — changed row/column values to hex
- op-desc.mif
  - — created new temporary file to begin documenting new and changed opcode formats/descriptions/(maybe pseudocode)
- xlu.mif
  - — additional information on the group size for E instructions
- memory.mif
  - — Changes to executable code space column in memory map
  - — some corrections to the memory map and unused space tables
  - — added section for watchdog timer
  - — updated NB connections diagram
- reset.mif
  - — More information on power up and resets, clears, etc.

MU 0054565

- cerberus.mif
  - some minor formatting changes
  - added note to reg. 6 that icache/dcahce configuration take effect immediately
  - refresh enable in octlet 10 powers up at 0.
  - added note to knob settings that reset for over-temp condition overrides default values to 1 (minimum power)
  - corrected reg. 30 global resistor setting from 0x1110 to 7
  - changed reg. 31 PLL1 and PLL0 divide ratio value defaults

List of changes: 1.2 - 1.3

Some small changes throughout book. Also:

- opcodes.mif
  - eliminated extract table after major opcodes, redundant with Tom's extract table
- xlu.mif
  - correction to bit field withdraw/deposit exception condition
- memory.mif
  - many small changes throughout chapter
  - added column to memory map saying which spaces code can execute from.
  - document that there's no unused space in ROM address range
  - updated gtlb, dcache tag, and icache tag protection fields
  - updated LTLB and address translation info due to change in value of LTLB mask field
  - corrected definition of exception 10
- cerberus.mif
  - register 6, NB priority, change default value to 16
  - registers 25-29, unused fields power up to 0
  - register 30 global resistor setting powers up to 1110

List of changes: 1.1(external) - 1.2

- intro.mif
  - formatting/terms/typo stuff
- opcodes.mif
  - added byte order to opcode structure
  - added minor opcode structure
- xlu.mif: Bit Field Withdraw/Deposit:
  - changed x to s/u in the gextracti/guextracti encoding table
  - corrected inconsistency in imm1 and imm2 definition
- pipelines.mif
  - formatting/grammar
- memory.mif:
  - general formatting and terms
  - converted space field on memory map to hex
  - added Access field to memory map
  - reworded some descriptions in unused space table and updated some entries
  - Added to DRAM description
  - Moved address translation description after GTLB/LTLB
  - Corrected address translation formula to account for privilege
  - added descriptions for new exceptions
  - split up data and instruction cache tag information
  - updates to NB description and connection diagram
- events.mif
  - formatting/terms
- cerberus.mif
  - added a few cross-references to other chapters and TSA